On the Effect of Dropping Layers of Pre-trained Transformer Models

Hassan Sajjad[♣1] Fahim Dalvi[♢] Nadir Durrani[♢] Preslav Nakov[♠1]
[♣]Faculty of Computer Science, Dalhousie University, Canada
[♢]Qatar Computing Research Institute, Hamad Bin Khalifa University, Qatar
[♠]Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE
hsajjad@dal.ca,{faimaduddin, ndurrani}@hbku.edu.qa, preslav.nakov@mbzuai.ac.ae

Abstract

Transformer-based NLP models are trained using billions of parameters, limiting their applicability in computationally constrained environments. While the number of parameters generally correlates with performance, it is not clear whether the entire network is required for a downstream task. Motivated by the recent work on pruning and distilling pre-trained models, we explore strategies to drop layers in pre-trained models, and observe the effect of pruning on downstream GLUE tasks. We were able to prune BERT, RoBERTa and XLNet models up to 40%, while maintaining up to 98% of their original performance. Additionally we show that our pruned models are at par with those built using knowledge distillation, both in terms of size and performance. Our experiments yield interesting observations such as: (i) the lower layers are most critical to maintain downstream task performance, (ii) some tasks such as paraphrase detection and sentence similarity are more robust to the dropping of layers, and (iii) models trained using different objective function exhibit different learning patterns and w.r.t the layer dropping.[1]

Keywords: pre-trained transformer models, efficient transfer learning, interpretation and analysis

[1]The work was done while the author was at QCRI
[1]The code is available at `https://github.com/hsajjad/transformers/`.

1. Introduction

Pre-trained Transformer models have achieved state-of-the-art performance on natural language processing tasks and have been adopted as feature extractors for solving downstream tasks such as question answering, natural language inference, and sentiment analysis. The current state-of-the-art Transformer-based pre-trained models consist of dozens of layers and millions of parameters. While deeper and wider models yield better performance, they also need large GPU/TPU memory. For example, BERT-large [1] is trained with 335 million parameters, and requires at least 24 GB of GPU memory to load. The larger size of these models limits their applicability in time- and memory-constrained environments.

Several methods have been proposed to reduce the size of pre-trained models. Notable approaches include pruning parts of the network after training [2, 3, 4], reduction through weight factorization and sharing [5], compression via knowledge-distillation [6] and quantization [7, 8]. Our work falls under the class of pruning methods.

The central argument governing pruning methods is that deep neural models are over-parameterized and that not all parameters are strictly needed, especially at the inference time. For example, previous research has shown that most of the attention heads can be removed [9, 3] or reallocated [10] without significantly impacting performance. Gordon et al. [11] pruned the least important weights in the network. We build our work based on similar observations, but we are interested in (i) whether it is necessary to use all layers of a pre-trained model for downstream tasks, and if not, (ii) which layers are necessary to keep in order to maintain good task-specific performance while achieving efficiency in transfer learning.

Motivated by recent findings in representation learning, we propose novel strategies to drop layers in pre-trained models. Voita et al. [12] showed that the top layers are biased towards the pre-training objective, leading us to question whether they are necessary for downstream tasks. Michel et al. [9], Dalvi

et al. [13] discussed over-parameterization and the redundancy in pre-trained models, leading us to question whether adjacent layers contain redundant information. More concretely, we drop top, bottom, middle, or alternate layers in the network. We additionally present methods to find layers that contribute least in the network by using their activation patterns and weights. We apply our strategies on four state-of-the-art pre-trained models, BERT [1], RoBERTa [14], ALBERT [5] and XLNet [15]. The first three are auto-encoders, while XLNet is an auto-regressive model. ALBERT presents an interesting case in the mix as its layers share parameters. We additionally experiment using DistilBERT to analyze whether a distilled model can be pruned further. We evaluate against GLUE benchmark [16] a suite of language understanding tasks. Our findings are summarized below:

- We propose practical strategies to drop layers in pre-trained models for efficient transfer learning.
- We show that dropping top layers works consistently well across different tasks and pre-trained models, e.g., yielding 40% reduction in size while preserving up to 98.2% of the performance.
- Our reduced models perform on par with models built using knowledge distillation in terms of accuracy, model size and inference speed, without requiring costly training of a new model.
- One-third of a distilled models can also be pruned successfully with an average loss of 0.75 points
- Despite having cross-layer parameter sharing, ALBERT can still be pruned for efficient inference with a small drop in performance.
- Certain downstream tasks require as few 3 layers to maintain performance within 1% threshold.
- Comparing architectures, models show different learning dynamics. For example, compared to BERT, RoBERTa and XLNet learn task-specific

knowledge earlier in the network and are thus more robust to layer-dropping.

Contribution. While a number of studies partially overlap with the strategies and the findings presented in this work, this is the first work that thoroughly investigates the effect of various layer-dropping methods using a variety of pre-trained models and on a large number of tasks. We showed that i) models have different learning dynamics, ii) a smaller close to optimal network can be achieved by optimizing the number of layers to drop with respect to the task at hand, iii) a distilled model can also benefit from layer-dropping. Our work recommends to use top layer-dropping as an essential baseline when building distilled models. Moreover, it provides a cheap way to get smaller models of any architecture rapidly, that are both memory and speed efficient.

2. Related Work

Efficient Pre-trained Models: Work done on exploring methods to down-scale pre-trained models can be categorized into architecture-invariant compression [5, 17, 8], knowledge distillation [18, 6], and pruning [11, 19].

Quantization [7, 8], an *architecture-invariant* method, reduces the numerical precision of the weights of the model to fewer bits. *Knowledge distillation (KD)* also known as student-teacher model [20] trains a smaller model that mimics the behavior of the larger model. Researchers have experimented with learning from the outputs of the encoder layers [21, 22], from the output logits [6, 23], and from the attention maps [22, 24]. Another distinction is between general-purpose distilled models [6, 24] and task-specific ones [22, 25, 23, 21, 26].

Pruning methods involve removing some parts of the networks that are either redundant or less relevant to the task at hand. [11, 19, 27] pruned the least important weights in the network. Michel et al. [9], Voita et al. [3] demonstrated that most of the attention heads can be pruned at test time, which reduces the computation, and speeds up inference. Fan et al. [28] introduced *LayerDrop* during training that resulted in pre-trained models that are robust towards dropping of layers at inference time. Our work is similar to them as we also

remove layers from the network. But we show that layers can be dropped safely from the pre-trained models without the need for additional training using LayerDrop. Nevertheless our strategies can also be applied to a model trained using LayerDrop.

Recently, Peer et al. [29] proposed a greedy layer pruning method that drops layers based on their independent performance on the end task. Their assumption is that a local decision about a layer aligns with a globally correct selection of layers. We demonstrate that our results are comparable to theirs, but we need no additional training to find an optimal set of layers.

Sun et al. [21], Xu et al. [30] used the bottom six layers of the BERT-base model to initialize the student model. This is similar to one of our strategies. However, their performance is much lower compared to our method. Moreover, we provide a comprehensive evaluation of our strategies on four pre-trained models to prove their efficacy in reducing the size of the network.

Liu et al. [31], Schwartz et al. [32], Xin et al. [33], Zhou et al. [34] speed up the inference time by introducing dynamic exiting strategies. The limitation of their work are the memory footprints of the model that remain identical to the original model.

Representation analysis: A number of studies have analyzed representations of pre-trained models at layer-level and showed that they learn linguistic information [35, 36, 37, 38, 39, 40, 41, 42, 43, 44]. Belinkov et al. [45], Sajjad et al. [46] provided a comprehensive literature review of such work. While the representation analysis uncovers, what linguistic properties different layers capture, they do not reflect which layers are important for transfer learning to a downstream task. Recently, Tamkin et al. [47], Merchant et al. [48], Durrani et al. [49] attempted to address this by analyzing layer-wise transferability of features during fine-tuning. Tamkin et al. [47] reinitialized individual layers of pre-trained model and observed the effect on the fine-tuning performance. Merchant et al. [48] used probing classifier, layer-wise similarity and layer-ablation for their analysis. Our work is similar to their layer-ablation study which they carried out to understand the difficulty of a downstream task, but the premise of

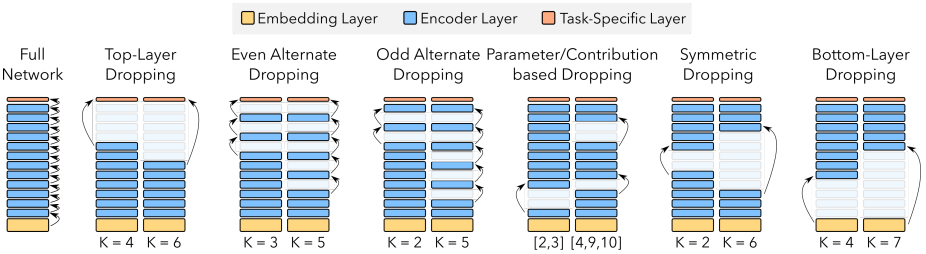


Figure 1: Illustration of layer-dropping strategies. K represents the number of layers that are dropped. For example, $K = 4$ in the top-layer strategy means top four layers of the model are dropped. In the contribution-based dropping, we select layers based on a similarity threshold. The number mentioned in the figure e.g. [2,3] shows the layers which are dropped based on the similarity threshold.

our work is very different. We gauge the importance of various subsets of layers with respect to the performance on downstream tasks, to achieve efficient models. Durrani et al. [49] used layer-wise and neuron probing classifiers [50, 51] and showed that core-linguistic knowledge is preserved in the lower layers of fine-tuned models. This resonates with our empirical finding that shows that higher layers can be safely pruned for efficient transfer learning.

3. Methodology

Consider a pre-trained language model $\mathbf{M}$ with an embedding layer E_0 and L encoder layers: $\{l_1, l_2, \ldots, l_L\}$. We probe whether it is necessary to keep all layers of the network for downstream tasks. We explore six strategies, that we describe below (also shown in Figure 1), to drop encoder layers from the model. Each pruning regime is followed by task-specific fine-tuning to analyze the effect of layer-dropping on the performance of the task.

3.1. Top-Layer Dropping

The top layers in pre-trained models are specialized towards the underlying objective function [12]. Zhang et al. [52] reinitialized the upper layers when fine-tuning towards GLUE task. We hypothesize that the top layers may not be

important when fine-tuning towards the a downstream task. In this strategy, we drop top K layers from the model. The output of layer l_{L-K} serves as the last layer of the reduced network. Then, a task-specific layer is added on top of this layer to perform task-specific fine-tuning. Figure 1 shows an example with dropping top 4 and 6 layers.

3.2. Alternate Dropping

Deep neural networks are innately redundant. Sun et al. [21] and Jiao et al. [22] amalgamated information from adjacent layers of the teacher model into a single layer of the student model. We hypothesize that neighbouring layers preserve similar information and may be dropped safely without any substantial loss of information. We drop N alternating odd or even layers from top to bottom of the network. For example in a 12-layer model with $K = 4$, we consider two sets of alternate layers: *Odd-alternate Dropping* – {5,7,9,11} and *Even-alternate Dropping* – {6,8,10,12}, see Figure 1 for illustration. When dropping an in-between layer l_i, the output of the previous layer l_{i-1} becomes the input of the next layer l_{i+1}, causing a mismatch in the expected input to l_{i+1}. However, we hope that during task-specific fine-tuning, the model will recover from this discrepancy.

3.3. Parameter-Based Dropping

In this approach, we estimate the importance of a given layer based on the model parameters. More specifically, we rank the layers based on their weights. We tested two hypotheses: (i) higher *magnitude* of the weights signals higher layer importance, (ii) higher *variance* of the weights corresponds to higher layer importance. We refer to the former as `Aggregation Method`, where we aggregate the weights of a layer, and we call the latter a `Variance Method`, where we calculate the variance of each layer. We drop the layers with the lowest aggregation or variance scores. Note that a transformer block has various sub-layers, but in our experiments we only used the final weights. We leave experiments with other layers within a transformer block as a possible direction for future work.

3.4. Contribution-Based Dropping

Our next strategy is based on the idea that a layer contributing below a certain threshold might be a good candidate for dropping. We define the contribution of a layer l_i in terms of the cosine similarity between its input and its output representations. A layer l_i with a high similarity (above a certain threshold) indicates that its output has not changed much from its input, and therefore it can be dropped from the network. More concretely, in the forward pass, we calculate the cosine similarity between the representation of the sentence token (CLS) before and after each layer. We average the similarity scores of each layer over the development set, and select layers that have an average similarity above a certain threshold for dropping. This *contribution-based strategy* can be seen as a principled variation of *alternate dropping*.

3.5. Symmetric Dropping

The bottom layers are closer to the input while the top layers are closer to the output. It is possible that both the top layers and the bottom layers are more important than the middle layers. The *Symmetric dropping* strategy retains the top and the bottom X layers, and drop K middle layers, where $2X + K = L$. For example, in a 12-layer model, if $K = 6$, we retain three top and three bottom layers, dropping layers 4–9. The output of layer 3 would then serve as an input to layer 10.

3.6. Bottom-Layer Dropping

Previous work on analyzing layers in Neural Networks [35, 41, 39, 53, 54] has shown that the lower layers model local interactions between words (which is important for morphology and lexical semantics), thus providing essential input to the higher layers. Removing lower layers could be therefore catastrophic. We still perform these experiments for the sake of completeness. We remove the bottom K layers of the model. The output of the embedding layer l_0 serves as an input to layer l_{K+1} of the original model.

Task	Description	Train	Dev
SST-2	Sentiment analysis	67349	872
MRPC	Microsoft Research paraphrase corpus	3668	408
MNLI	Natural language inference	392702	9815
QNLI	Question natural language inference	104743	5463
QQP	Quora question pairs	363846	40430
RTE	Recognizing textual entailment	2490	277
STS-B	Semantic textual similarity	5749	1500

Table 1: Data statistics of the GLUE tasks. All tasks are binary classification tasks, except for STS-B which is a regression task. Recall that the test sets are not publicly available, and hence we use development set to report results.

4. Experimental Setup

Datasets. We evaluated our strategies on General Language Understanding Evaluation (GLUE) tasks [16] tasks, which serves as a defacto standard to evaluate pre-trained language models. Table 1 provides statistics of each dataset. More specifically, we evaluated on the following tasks: SST-2 for sentiment analysis with the Stanford sentiment treebank [55], MNLI for natural language inference [56], QNLI for Question NLI [57], QQP for Quora Question Pairs,[2] RTE for recognizing textual entailment [58], MRPC for Microsoft Research paraphrase corpus [59], and STS-B for the semantic textual similarity benchmark [60]. We left out WNLI, due to the irregularities in its dataset, as also reported by others,[3] as well as CoLA due to large variance and unstable results across fine-tuning runs.

Models. We experimented with three state-of-the-art 12-layered pre-trained models [4] BERT [1], RoBERTa [14] and XLNet [15]. We additionally experimented using a 12-layered ALBERT [5] model and a distilled model, DistilBERT [6].

[2] `http://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`

[3] `http://gluebenchmark.com/faq`

[4] For the sake of clarity when the trends are similar across models, we present the results of selected models only.

Our selection of models encourage interesting comparison between different types of models such as auto-regressive vs. auto-encoder and a large model vs. its distilled version. All experiments are conducted using the `transformers` library [61]. We used the default settings and did not optimize the parameters. We limit our experiments to the base versions of the transformers as we could not experiment with BERT-large or XLNet-large due to memory limitations.[5] However, our strategies are straightforward to apply to models of any depth.

End-to-End Procedure. Given a pre-trained model, we drop layers using one of the strategies described in Section 3. We then performed task-specific fine-tuning using GLUE training sets for three epochs as prescribed by [1][6] and evaluated on the official devsets.

5. Evaluation Results

We experimented with dropping K number of layers where $K = 2, 4, 6$ in BERT, RoBERTa and XLNet, and $K = 1, 2, 3$ in DistilBERT (a 6-layer model). As an example, for $K = 2$ on a 12-layer model, we drop the following layers: top strategy – $\{11, 12\}$; bottom strategy – $\{1, 2\}$; even-alternate – $\{10, 12\}$; odd-alternate – $\{9, 11\}$; symmetric – $\{6, 7\}$. For the parameter-based strategy, we calculate the score of every layer based on the aggregated weights and the variance in the weights, and we drop the layers with the lowest score. In the contribution-based strategy, the dropping of layers is dependent on a similarity threshold. We calculate the similarity between input and output of each layer and remove layers with similarity above the threshold values of 0.95, 0.925 and 0.9. These values were chosen empirically. A threshold value below 0.9 or above

[5] In order to fit large models in our TitanX 12GB GPU cards, we tried to reduce the batch size, but this yielded poor performance, see `https://github.com/google-research/bert#out-of-memory-issues`.

[6] We experimented with using more epochs, especially for dropping strategies that exclude in-between layers, in order to let the weight matrix adapt to the changes. However, we did not see any benefit in going beyond three epochs.

0.95 resulted in either more than half of the network being considered as similar, or none of the layers to be similar.

5.1. Comparing Strategies

Figure 2 presents average classification performance of BERT and XLNet using various layer-dropping strategies. We observe similar trends for RoBERTa and DistilBERT and limit the presentation of results to two models here.

Top-layer dropping consistently outperforms other strategies when dropping 6 layers. We dropped half of the top layers (yellow bars in the top strategy) with an average loss of only 2.91 and 1.81 points for BERT and XLNet respectively. The *Bottom-layer dropping* strategy performed the worst across all models, as expected, showing that it is more damaging to remove information from the lower layers of the network. The behavior of top and bottom dropping is consistent across all models. It nicely connects with findings in representation learning, i.e. lower layers learn core-linguistic phenomena and our results show that they are important to maintain task-specific performance.

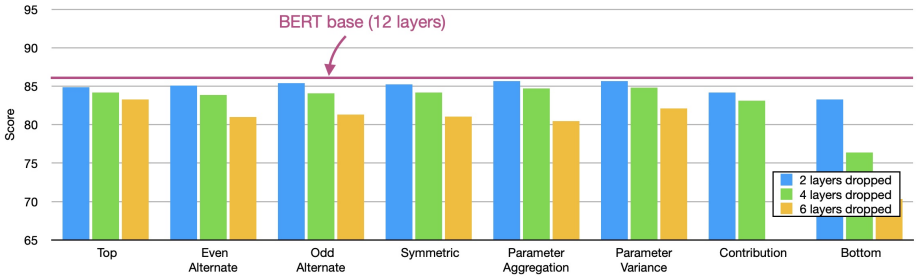
Parameter-based strategy using variance is the second best strategy at $K = 6$. Compared to most of the other strategies presented in this work, the parameter-based strategy makes a more informed decision based on the parameters of the model, i.e., the weights. We found the variance-based strategy to outperform the aggregation-based one, and thus we limit our discussion to the former only. The variance-based method selected different layers to drop for each model. The order of the six layers to drop is $\{1, 12, 8, 9, 11, 2\}$ for BERT, $\{11, 12, 6, 7, 5, 10\}$ for RoBERTa and $\{11, 12, 7, 8, 9, 10\}$ for XLNet. One common observation here is that the last 2–3 layers and the middle layers of the models can be removed safely with a small drop in performance (see the results of the variance-based method in Figure 2). Moreover, BERT is an exception where the first two contextualized layers $\{1, 2\}$ are also selected to be removed. This resulted in a huge loss in performance (see the results for BERT when dropping 6 layers based on the variance-based method). Interestingly, dropping 6-layers of XLNet resulted in a model that was identical to that of the top-layer

strategy, i.e., removing the top-6 layers. RoBERTa presents an interesting case where the parameter-based strategy resulted in a drop of the middle layers and of the top layers, while keeping the lower and the higher middle layers. The average results for RoBERTa when using the variance-based method are lower by 0.73 point only compared to the top-layer method. The promising results of the parameter-based method on two out of three models show its efficacy. Note that our current exploration is limited to the parameters of the base models. Fine-tuning substantially changes the parameters [49], which may result in a task-wise informed dropping of layers. We did not try task-specific pruning as the focus of our work is on task-agnostic efficient models.

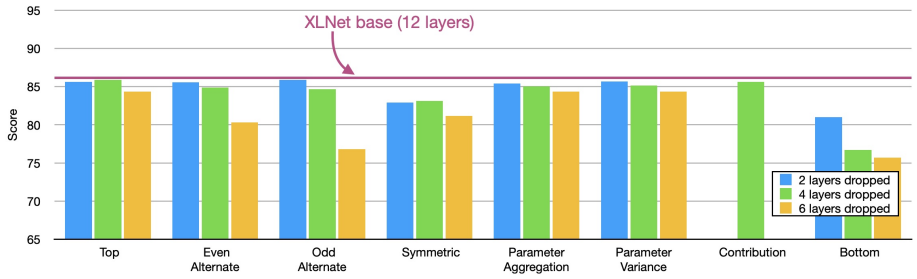
Dropping top alternate layers is better than dropping top consecutive layers. The *Odd-alternate dropping* strategy gave better results than the top at $K = 2$ (blue bars in the Odd-alternate strategy), across all the tasks. Looking at the layers that were dropped: top – $\{11, 12\}$; even-alternate – $\{10, 12\}$; odd-alternate – $\{9, 11\}$, we can say that (i) dropping last two consecutive layers $\{11, 12\}$ is more harmful than removing alternate layers, and (ii) keeping the last layer $\{9, 11\}$ is more important than keeping the second last layer with its alternate pair. At $K = 6$, the *Alternate dropping* strategies show a large drop in the performance, perhaps due to removal of lower layers. Recall that our results from the bottom strategy showed lower layers to be critical for transfer learning.

The *Symmetric strategy* gives importance to both top and bottom layers and drops the middle layers. Dropping two middle layers from BERT degrades the performance by 0.97 points and makes it the second best strategy at $K = 2$. However, on XLNet the performance degrades drastically when dropping the same set of layers. Comparing these two models, XLNet is sensitive to the dropping of middle layers while BERT shows competitive results to the *Top-layer dropping* strategy even after removing 4 middle layers. We analyze the difference in the behavior of models in Section 6.

For *Contribution-based strategy*, we chose layers $\{3, 5\}$ at threshold 0.95 and $\{3, 5, 8, 9\}$ at threshold 0.925 for BERT, and layers $\{9, 10, 11\}$ at threshold 0.925



(a) BERT



(b) XLNet

Figure 2: Average classification performance on GLUE tasks when using different layer-dropping strategies and when removing different numbers of layers for BERT and XLNet. Note that the contribution-based strategy selects layers based on the similarity threshold. In some cases it does not select (2,4 or 6) number of layers, which results in some missing bars in the figure. The horizontal red line represents the results using the full model.

and $\{8, 9, 10, 11\}$ at threshold 0.9 for XLNet. Using a lower or a higher similarity threshold resulted in dropping none or more than half of the layers in the network respectively. For BERT, the contribution-based dropping did not work well since the method chose a few lower layers for dropping. On the contrary, it worked quite well on XLNet where higher layers were selected. This is in-line with the findings of top and bottom strategy that all models are robust to dropping of higher layers compared to dropping of lower layers.

The contribution-based strategy is based on the activations of each layer, which is an input-dependent process. Depending on the nature of the input or the task, the activation patterns will change. We suspect that this is one of the reasons for the failure of the strategy. A strategy based on task-specific

Drop.	SST-2	MNLI	QNLI	QQP	STS-B	RTE	MRPC
				BERT			
0/12	92.43	84.04	91.12	91.07	88.79	67.87	87.99
2/12	92.20 (0.23↓)	83.26 (0.78↓)	89.84 (1.28↓)	90.92 (0.15↓)	88.70 (0.09↓)	62.82 (5.05↓)	86.27 (1.72↓)
4/12	90.60 (1.83↓)	82.51 (1.53↓)	89.68 (1.44↓)	90.63 (0.44↓)	88.64 (0.15↓)	67.87 (0.00)	79.41 (8.58↓)
6/12	90.25 (2.18↓)	81.13 (2.91↓)	87.63 (3.49↓)	90.35 (0.72↓)	88.45 (0.34↓)	64.98 (2.89↓)	80.15 (7.84↓)
				RoBERTa			
0/12	92.20	86.44	91.73	90.48	89.87	68.95	88.48
2/12	93.46 (1.26↑)	86.53 (0.09↑)	91.23 (0.50↓)	91.02 (0.54↑)	90.21 (0.34↑)	71.84 (2.89↑)	89.71 (1.23↑)
4/12	93.00 (0.80↑)	86.20 (0.24↓)	90.57 (1.16↓)	91.12 (0.64↑)	89.77 (0.10↓)	70.40 (1.45↑)	87.50 (0.98↓)
6/12	91.97 (0.23↓)	84.44 (2.00↓)	90.00 (1.73↓)	90.91 (0.43↑)	88.92 (0.95↓)	64.62 (4.33↓)	85.78 (2.70↓)
				XLNET			
0/12	93.92	85.97	90.35	90.55	88.01	65.70	88.48
2/12	93.35 (0.57↓)	85.67 (0.30↓)	89.35 (1.00↓)	90.69 (0.14↑)	87.59 (0.42↓)	66.06 (0.36↑)	86.52 (1.96↓)
4/12	92.78 (1.14↓)	85.46 (0.51↓)	89.51 (0.84↓)	90.75 (0.20↑)	87.74 (0.27↓)	67.87 (2.17↑)	87.25 (1.23↓)
6/12	92.20 (1.72↓)	83.48 (2.49↓)	88.03 (2.32↓)	90.62 (0.07↑)	87.45 (0.56↓)	65.70 (0.00)	82.84 (5.64↓)
				DistilBERT			
0/6	90.37	81.78	88.98	90.40	87.14	60.29	85.05
1/6	90.37 (0.00)	80.41 (1.37↓)	88.50 (0.48↓)	90.33 (0.07↓)	86.21 (0.93↓)	59.93 (0.36↓)	84.80 (0.25↓)
2/6	90.25 (0.12↓)	79.41 (2.37↓)	86.60 (2.38↓)	90.19 (0.21↓)	86.91 (0.23↓)	62.82 (2.53↑)	82.60 (2.45↓)
3/6	87.50 (2.87↓)	77.07 (4.71↓)	85.78 (3.20↓)	89.59 (0.81↓)	85.19 (1.95↓)	58.48 (1.81↓)	77.45 (7.60↓)

Table 2: Task-wise performance for the top-layer dropping strategy using the official GLUE development sets. Drop. represents the number of layers that are dropped in comparison to the total number of layers in the model. The red numbers with downward arrow shows the drop in performance in comparison to using the full model i.e. 0/12 and the blue numbers with upward arrow shows the gain in performance.

contribution might yield a better performance. However, in this work we focused on task-independent efficient models, leaving task-dependent models for future work.

5.2. Task-wise Results

Top-layer strategy works consistently well for all models at $K = 6$. In the rest of the paper, we discuss the results for the *Top-layer strategy* only, unless specified otherwise. Table 2[7] presents the results for the individual GLUE tasks

[7]We use default settings provided in the Transformer library. This causes a slight mismatch between some numbers mentioned in the original papers of each models and our paper.

using the *Top-layer strategy* on three pre-trained models and a distilled model. We observe the same trend as for the averaged results: for most of the tasks, we can safely drop half of the top layers in BERT, RoBERTa and XLNet losing only 1-3 points.

The paraphrase task (QQP) and sentence similarity task (STS-B) are least affected by the dropping of layers. When dropping half of the layers, there was no loss in performance for QQP on XLNet and RoBERTa, and a loss of 0.72 only for BERT. Similarly, for STS-B we observed a decrease of only 0.56, 0.95 and 0.34 points for XLNet, RoBERTa and BERT respectively. In contrast, RTE and MRPC tasks show substantial change (gain/drop) in the performance with layer-dropping when compared with using the full model (see BERT and RoBERTa 0/12,2/12,4/12 results). This is due to the small size of the dev sets, 408 and 277 instances for MRPC and RTE respectively. A few right and wrong predictions cause a large variation in the overall score. We use McNemar's test at p=value=0.05, and we found these differences, such as 5.05 points drop in the performance of BERT for RTE, statistically insignificant.

Dropping top two layers for RoBERTa resulted in better performance and stability. Interestingly, in several cases for RoBERTa, dropping two layers resulted in *better* performance than using the full model. Moreover, we observed that layer-dropping resulted in stable runs and was less prone to initialization seed and batch size. We used default settings for all the model and did not investigate the effect of parameter optimization on the performance of the pre-trained and reduced models to have comparable results.

A distilled model can also be pruned successfully. We observed a similar trend, dropping layers in DistilBERT compared to BERT model. It is interesting to see that an already distilled version of the model can be further pruned by a third, with an average loss of 0.75 points only. However, dropping half of its layers drastically degrades the performance on several tasks. Schwartz et al. [32] also showed that pruning is orthogonal to model distillation.

5.3. Memory and Speed Comparison

Dropping layers reduces the number of parameters in the network, significantly speeding up the task-specific fine-tuning and the inference time. Table 3 compares the number of parameters, and the speed up in the fine-tuning and decoding time, versus the loss in performance. We see that dropping top half of the layers of the network, reduced the number of parameters by 40%, speeding up fine-tuning and inference by 50% with average performance loss between 0.89–2.91 points. The results for RoBERTa are even remarkable; as with all the memory and speed improvements, the average performance dropped by only 0.89 points. Dropping 4 layers (which gives a speed-up of 33%), RoBERTa achieved a performance close to dropping no layers. XLNet also showed robustness to the drop of top 4 layers and the performance dropped by only 0.23 points. It is worth noting that a better trade-off between computational efficiency and loss in performance can be achieved by optimizing for a specific task. For example QQP maintained performance within 1% on XLNet when 9 layers were dropped (See Table 4). This corresponds to 60% reduction in the number of parameters and 80% reduction in terms of inference time.

6. Discussion

Now we perform further analysis and discuss variations of our methodology. We limit the results to 5 most stable tasks (SST-2, MNLI, QNLI, QQP, STS-B).

6.1. Task-specific optimal number of layers to drop.

The variation in the amount of loss for each task with the dropping of layers in Table 2 suggests that the task-specific optimal number of layers would result in a better balance between the size of the pruned model and the loss in performance. In this section, we present the results of the optimal number of layers for each task. For these experiments, we split the standard development set into equal-sized hold-out set and dev set. We find the minimum number of layers required to maintain 1%, 2%, and 3% performance on the dev set using

Drop.	Loss	Param.	Fine-tuning speedup	Inference seconds
	BERT ‖ RoBERTa			
0/12	0.00 ‖ 0.00	110M	1.00	-
2/12	1.33 ‖ -0.42	94M	1.24	17% ↓
4/12	2.00 ‖ 0.01	80M	1.48	33% ↓
6/12	2.91 ‖ 0.89	66M	1.94	50% ↓
	XLNET			
0/12	0.00	116M	1.00	-
2/12	0.54	101M	1.20	16% ↓
4/12	0.23	86M	1.49	32% ↓
6/12	1.81	71M	1.96	49% ↓

Table 3: Comparing the number of parameters (Param.), the speed up in the fine-tuning step, and the inference time for different models. *Fine-tuning speedup* shows how many times the model speeds up compared to the original network. We report inference time on the QQP devset consisting of 40.4k instances with a batch size of 32.

our top-layer strategy and we verify that the findings generalize to the hold-out test. Table 4 shows the optimal number of layers on dev and the corresponding percentage of performance drop on the hold-out set (in parentheses). For most of the cases, the optimal number of layers found using the dev set aligns well with the hold-out set. For example, BERT QNLI with 1% loss in performance showed that one layer can be dropped safely and this results in a loss of 0.84 points absolute compared to using the full model.

Overall, RoBERTa and XLNet showed most robustness towards the dropping of layers while maintaining performance threshold of 1%. For example, QQP maintained performance within 1 point even when the top 9 and 8 layers of XLNet and RoBERTa respectively were dropped. Essentially, the model consists of only three layers – $\{1, 2, 3\}$. On the contrary, dropping 9 layers in BERT resulted in a loss of 3% points for the QQP task.

	SST-2	MNLI	QNLI	QQP	STS-B
			1% Loss Threshold		
BERT	7(1.6)	3(1.04)	1(0.84)	6(0.75)	7(1.16)
RoBERTa	4(0.00)	4(0.20)	5(0.87)	8(0.77)	5(1.22)
XLNet	8(1.38)	5(1.22)	4(0.51)	9(0.60)	7(0.05)
			2% Loss Threshold		
BERT	7(1.60)	5(1.26)	3(1.68)	8(1.60)	7(1.16)
RoBERTa	4(0.00)	5(1.26)	6(1.42)	9(1.51)	6(2.31)
XLNet	8(1.38)	5(1.22)	6(1.46)	9(0.60)	8(1.22)
			3% Loss Threshold		
BERT	8(2.06)	6(2.42)	5(2.60)	9(2.27)	8(2.61)
RoBERTa	5(0.69)	6(2.73)	7(2.37)	10(3.21)	7(3.00)
XLNet	8(1.38)	6(1.55)	7(1.61)	9(0.60)	9(2.46)

Table 4: Number of layers dropped from the network while maintaining performance within a pre-defined threshold. The numbers outside brackets are the optimal number of layers found using the dev set and the numbers within brackets report the performance loss on the hold-out set. For example in 7(1.6), 7 are the optimal number of layers that can be dropped based on the dev set and 1.6 is the performance loss when 7 layers are dropped on the hold-out set.

6.2. Comparing Pre-trained Models

Our pruning strategies illuminate model-specific peculiarities that help us in comparing and understanding the learning dynamics of these models. **RoBERTa and XLNet learn task-specific knowledge earlier in the network compared to BERT**. Figure 3 shows the average layer-wise performance of each model. RoBERTa learns task-level information much earlier in the model (see the steep slope of the yellow line for lower layers). Although XLNet starts similar to BERT but in the lower-middle layers, it learns the task information relatively faster than BERT. For both RoBERTa and XLNet, the performance matures close to the 7^{th} layer of the model while BERT improves with every higher layer until the 11^{th} layer. Since XLNet and RoBERTa mature much earlier in the network, this suggests that top layers in these networks might be redundant for downstream tasks and are a good candidate for dropping in exchange for a small

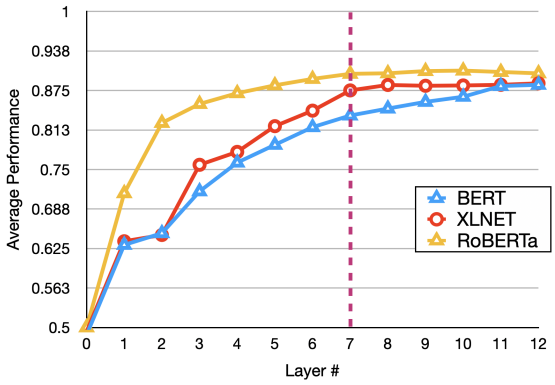


Figure 3: Average layer-wise classification results.

loss in performance. This observation is in line with the results presented in Table 2. For example, we showed that the drop of top two layers of RoBERTa resulted in either marginal drop in performance or improvement in performance.

The difference between the learning dynamics of BERT and RoBERTa encourages further investigation into what caused RoBERTa to learn task-specific knowledge earlier in the network. Is it because of the large amount of training data used for RoBERTa or because of better pre-training procedures such as dynamic masking, and exclusion of next sentence prediction loss? Does early learning of task-specific knowledge as in XLNet and RoBERTa reflect towards a better and robust pre-trained model? Answering these questions is important for improving the design of pre-trained models and require future exploration.

6.3. Pruning the ALBERT Model

ALBERT is based on the cross-layer parameter sharing. Because of this, our layer dropping strategies do not save any memory as opposed to using BERT and other transformers. However, it still makes the inference faster by speeding up the forward pass. Table 5 presents the results on five GLUE tasks. Interestingly, dropping the top-6 layers did not result in drastic degradation of the model performance and, in some cases, the results even improved compared to using the baseline model. For example, in the case of SST-2, the performance of a

Drop	SST-2	MNLI	QNLI	QQP	STS-B
0/12	89.79	83.39	**90.24**	**90.29**	**89.61**
2/12	91.40	**83.82**	89.55	89.64	89.54
4/12	**91.63**	82.73	**90.24**	88.51	87.00
6/12	90.14	81.64	89.11	90.08	88.21

Table 5: ALBERT: task-wise performance for the top-layer dropping strategy using the official GLUE dev-sets. **Drop** shows the number of layer dropped/the total layers in the model.

6-layered model is 90.14, which is 0.35 points absolute better than the baseline. Compared to the 6-layered BERT model (Table 2), the drop in the performance of ALBERT-6 is relatively small. We hypothesize that the parameter sharing in the case of ALBERT make the model learn much richer representation in the shared contextualized layers of the model, which yields a model that is robust towards layer-dropping. These results are encouraging and show that the model that was designed to be space-efficient can be further improved towards run-time efficiency by simply pruning some of its layers.

6.4. Comparing against Distilled Models

We now compare the performance of our pruned models when applying the top-layer dropping strategy to distilled and pruned models built using various sophisticated architectures and training procedures. In particular, we compare to previous work [6, 25, 23] that used KD to build 6-layered distilled models. More specifically, we present the result of the following distilled models; Vanilla-KD – a distilled model built using the original KD method [18], BERT-PKD [21] – patient knowledge distillation method that encourages a student model to learn from various layers of the teacher model, and BERT-TH – a theseus compression method that gradually distill layers of a large model. Additionally, we compare with the pruned RoBERTa model of [28] that used layer-level dropout during training of a pre-trained model and showed that it enables robust dropping of layers at test time. We also compare to the greedy layer pruning method [29],

which creates task-specific smaller-size models by dropping layers in a greedy fashion. All these models are identical in size to our smaller models obtained by dropping the top-6 layers in BERT and RoBERTa. We refer to them as *BERT-6* and *RoBERTa-6*. Table 6 compares the results.[8]

Our pruned models (BERT-6 and RoBERTa-6) showed competitive performance compared to their distilled versions built using KD. This result is quite surprising, given that our pruned models do not require any additional training, while building a distilled model using KD requires training from scratch, which is a time consuming and computation expensive process. The top layer-dropping works consistently for all model types including distilled models and a large set of language understanding tasks. Moreover, our setup offers the flexibility to choose different sizes of the model based on the computational requirements and the specifics of a downstream task. The result of preserving bottom layers of the model suggests selective compression applied to pre-trained models. For example, in KD while combining information from various layers of the large model, it is advisable to preserve the bottom layers and distilled the top layers. Similarly, pruning methods such as weight and attention-head pruning, and quantization can be aggressively applied to top layers of the models while preserving the bottom layers.

Our RoBERTa-6 has comparable results to the 6-layer pruned model trained using LayerDrop and Greedy layer pruning. Fan et al. [28] used layer-level dropout during training of a pre-trained model and showed that it enables robust dropping of layers at test time. Similar to us, they directly pruned top 6-layers of their large model and fine-tuned it for specific tasks. Table 6 (row 7 and 10) compares top-layer dropping using their model and the original RoBERTa model. On two out of three tasks, dropping top-layers from the original RoBERTa model outperformed training a new model using LayerDrop. This shows that the current models are already robust and the top-layer

[8]There is an exhaustive list of task-specific distilled models but we show the results for a few for comparison.

No.	Model	SST-2	MNLI	QNLI	QQP	STS-B
1.	Vanilla-KD	90.50	80.10	88.00	88.10	84.90
2.	BERT-PKD	91.30	81.30	88.40	88.40	86.20
3.	BERT-TH	**91.80**	**82.10**	88.80	88.80	87.80
4.	GLP_6	91.20	81.30	87.60	86.80	87.60
5.	DistilBERT	90.37	81.78	**88.98**	**90.40**	87.14
6.	BERT-6	90.25	81.13	87.63	**90.35**	**88.45**
7.	Fan et al. RoBERTa-6	92.50	82.90	89.40	-	-
8.	GLP_6	92.00	**85.60**	**90.80**	87.80	86.60
9.	DistilRoBERTa	**92.50**	84.00	**90.80**	89.40	88.30
10.	RoBERTa-6	91.97	84.44	90.00	**90.91**	**88.92**

Table 6: Comparing 6-layered BERT and RoBERTa models. Results of Vanilla-KD, BERT-PKD and BERT-TH are taken from Xu et al. [30]. Fan et al. results and GLP_6 are taken from [28, 29]. BERT-6 and RoBERTa-6 represent our models achieved by pruning top 6 layers.

dropping strategy can be directly applied to the available pre-trained models. Similarly, we found that despite optimizing the model towards a downstream GLUE task, the greedy layer pruning (GLP_6) did not show a clear advantage over our 6-layered model. For example, compared to BERT (rows 4 and 6), our BERT-6 model yields better or comparable performance to GLP_6 on the QQP, STS-B, MNLI and QNLI tasks, and performs worse only on the SST-2 task.

6.5. Layer-Dropping using Fine-tuned Models

Here, we question whether dropping layers from a fine-tuned model is more effective than dropping them from a base model? To answer this, we first fine-tune the model, drop the layers, and then fine-tune the reduced model again. Table 7 presents the results on BERT and XLNet. We found this setup to be comparable to dropping layers directly from the pre-trained model in most of the cases. This shows that dropping top layers directly from a pre-trained model does not lose any critical information which was essential for a specific task. However, we think that pruning a fine-tuned model may lose task-specific information because the model is optimized for the task. Dropping layers may

Model	SST-2	MNLI	QNLI	QQP	STS-B
BERT-6	**92.25**	**81.13**	**87.63**	90.35	**88.45**
BERT-FT-6	90.02	80.85	87.24	90.34	88.16
XLNet-6	92.20	83.48	**88.03**	90.62	87.45
XLNet-FT-6	**92.43**	**83.75**	86.80	**90.77**	**87.60**

Table 7: Layer-dropping using task-specific models. XLNet-FT-6 first fine-tunes the pre-trained model, removes the layers and performs fine-tuning again.

have severe effect. This is reflected in some of the results of BERT-6.

Gradual Dropping:. In another attempt to preserve the model's performance during the dropping process, we iteratively drop one layer after every two epochs of the fine-tuning process. This did not yield any improvement over dropping layers directly from the model.

7. Conclusion

We proposed strategies to drop layers in pre-trained models and analyzed the model behavior on downstream tasks. We conducted experiments using a variety of pre-trained models and using a diverse set of natural language understanding tasks and showed that one can reduce the model size by up to 40%, while maintaining up to 98% of their original performance on downstream tasks. Our pruned models performed on par with distilled models building using knowledge distillation. However, unlike distilled models, our approach does not require re-training, is applicable to a large set of pre-trained models including distilled models, and provides the flexibility to balance the trade-off between accuracy and model size. Moreover, we made several interesting observations such as, i) the lower layers are most critical to maintain downstream task performance, ii) certain downstream tasks require as few as only 3 layers out of 12 layers to maintain within 1% performance threshold, iii) networks trained using different objective functions have different learning patterns e.g. XLNet and RoBERTa learns task-specific information much earlier in the network compared to BERT.

References

[1] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, 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, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019.

[2] P. Michel, O. Levy, G. Neubig, Are sixteen heads really better than one?, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 14014–14024. URL: http://papers.nips.cc/paper/9551-are-sixteen-heads-really-better-than-one.pdf.

[3] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, I. Titov, 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, Florence, Italy, 2019. doi:10.18653/v1/P19-1580.

[4] J. S. McCarley, Pruning a bert-based question answering model, 2019. arXiv:1910.06360.

[5] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, Albert: A lite bert for self-supervised learning of language representations, 2019. arXiv:1909.11942.

[6] V. Sanh, L. Debut, J. Chaumond, T. Wolf, Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019. arXiv:1910.01108.

[7] O. Zafrir, G. Boudoukh, P. Izsak, M. Wasserblat, Q8bert: Quantized 8bit bert, 2019. arXiv:1910.06188.

[8] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, K. Keutzer, Q-bert: Hessian based ultra low precision quantization of bert, 2019. arXiv:1909.05840.

[9] P. Michel, O. Levy, G. Neubig, Are sixteen heads really better than one?, CoRR abs/1905.10650 (2019). URL: http://arxiv.org/abs/1905.10650.

[10] H. Peng, R. Schwartz, D. Li, N. A. Smith, A mixture of $h - 1$ heads is better than h heads, 2020. arXiv:2005.06537.

[11] M. A. Gordon, K. Duh, N. Andrews, Compressing BERT: Studying the effects of weight pruning on transfer learning, ArXiv abs/2002.08307 (2019).

[12] E. Voita, R. Sennrich, I. Titov, The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives, 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), Hong Kong, China, 2019.

[13] F. Dalvi, H. Sajjad, N. Durrani, Y. Belinkov, Analyzing redundancy in pretrained transformer models, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Online, 2020, pp. 4908–4926. URL: https://aclanthology.org/2020.emnlp-main.398. doi:10.18653/v1/2020.emnlp-main.398.

[14] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized BERT pretraining approach, CoRR abs/1907.11692 (2019). URL: http://arxiv.org/abs/1907.11692. arXiv:1907.11692.

[15] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, Q. V. Le, Xlnet: Generalized autoregressive pretraining for language understanding, 2019. arXiv:1906.08237.

[16] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, 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, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 353–355. URL: https://www.aclweb.org/anthology/W18-5446. doi:10.18653/v1/W18-5446.

[17] Q. Cao, H. Trivedi, A. Balasubramanian, et al., Faster and just as accurate: A simple decomposition for transformer models, ICLR Openreview (2020).

[18] G. E. Hinton, S. Osindero, Y. W. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18 (2006) 1527–1554.

[19] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Machine Learning Research 3 (2003).

[20] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015. URL: http://arxiv.org/abs/1503.02531, cite arxiv:1503.02531Comment: NIPS 2014 Deep Learning Workshop.

[21] S. Sun, Y. Cheng, Z. Gan, J. Liu, Patient knowledge distillation for BERT model compression, 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), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 4322–4331. URL: https://www.aclweb.org/anthology/D19-1441. doi:10.18653/v1/D19-1441.

[22] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, Q. Liu, Tinybert: Distilling bert for natural language understanding, 2019. arXiv:1909.10351.

[23] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, J. Lin, Distilling task-specific knowledge from BERT into simple neural networks, CoRR abs/1903.12136 (2019). URL: http://arxiv.org/abs/1903.12136.

[24] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, D. Zhou, "mobilebert: Task-agnostic compression of bert by progressive knowledge transfer", in: In-

ternational Conference on Learning Representations, 2020. URL: https://openreview.net/forum?id=SJxjVaNKwB.

[25] I. Turc, M.-W. Chang, K. Lee, K. Toutanova, Well-read students learn better: On the importance of pre-training compact models, 2019. arXiv:1908.08962.

[26] H. Tsai, J. Riesa, M. Johnson, N. Arivazhagan, X. Li, A. Archer, Small and practical bert models for sequence labeling, 2019. arXiv:1909.00100.

[27] A. Renda, J. Frankle, M. Carbin, Comparing rewinding and fine-tuning in neural network pruning, in: ICLR, 2020.

[28] A. Fan, E. Grave, A. Joulin, Reducing transformer depth on demand with structured dropout, 2019. arXiv:1909.11556.

[29] D. Peer, S. Stabinger, S. Engl, A. J. Rodríguez-Sánchez, Greedy layer pruning: Decreasing inference time of transformer models, CoRR abs/2105.14839 (2021). URL: https://arxiv.org/abs/2105.14839. arXiv:2105.14839.

[30] C. Xu, W. Zhou, T. Ge, F. Wei, M. Zhou, Bert-of-theseus: Compressing bert by progressive module replacing, ArXiv abs/2002.02925 (2020).

[31] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, Q. Ju, Fastbert: a self-distilling bert with adaptive inference time, 2020. arXiv:2004.02178.

[32] R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, N. A. Smith, The right tool for the job: Matching model and instance complexities, 2020. arXiv:2004.07453.

[33] J. Xin, R. Tang, J. Lee, Y. Yu, J. Lin, Deebert: Dynamic early exiting for accelerating bert inference, 2020. arXiv:2004.12993.

[34] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, F. Wei, BERT loses patience: Fast and robust inference with early exit, 2020. arXiv:2006.04152.

[35] Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, J. Glass, What do Neural Machine Translation Models Learn about Morphology?, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Association for Computational Linguistics, Vancouver, 2017. URL: https://aclanthology.coli.uni-saarland.de/pdf/P/P17/P17-1080.pdf.

[36] F. Dalvi, N. Durrani, H. Sajjad, Y. Belinkov, S. Vogel, Understanding and improving morphological learning in the neural machine translation decoder, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017, pp. 142–151. URL: https://aclanthology.org/I17-1015.

[37] Y. Belinkov, L. Màrquez, H. Sajjad, N. Durrani, F. Dalvi, J. Glass, Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017, pp. 1–10. URL: https://aclanthology.org/I17-1001.

[38] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, M. Baroni, What you can cram into a single vector: Probing sentence embeddings for linguistic properties, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), 2018.

[39] N. F. Liu, M. Gardner, Y. Belinkov, M. E. Peters, N. A. Smith, Linguistic knowledge and transferability of contextual representations, 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), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 1073–1094. URL: https://www.aclweb.org/anthology/N19-1112.

[40] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. V. Durme, S. R. Bowman, D. Das, E. Pavlick, What do you learn from context? probing for sentence structure in contextualized word representations, 2019. arXiv:1905.06316.

[41] I. Tenney, D. Das, E. Pavlick, BERT rediscovers the classical NLP pipeline, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 4593–4601. URL: https://www.aclweb.org/anthology/P19-1452. doi:10.18653/v1/P19-1452.

[42] N. Durrani, F. Dalvi, H. Sajjad, Y. Belinkov, P. Nakov, One size does not fit all: Comparing NMT representations of different granularities, 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), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 1504–1516. URL: https://aclanthology.org/N19-1154. doi:10.18653/v1/N19-1154.

[43] Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, J. Glass, On the linguistic representational power of neural machine translation models, Computational Linguistics 46 (2020) 1–52. URL: https://aclanthology.org/2020.cl-1.1. doi:10.1162/coli_a_00367.

[44] D. Arps, Y. Samih, L. Kallmeyer, H. Sajjad, Probing for constituency structure in neural language models, 2022. doi:10.48550/ARXIV.2204.06201.

[45] Y. Belinkov, S. Gehrmann, E. Pavlick, Interpretability and analysis in neural NLP, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, Association for Computational Linguistics, Online, 2020, pp. 1–5. URL: https://aclanthology.org/2020.acl-tutorials.1. doi:10.18653/v1/2020.acl-tutorials.1.

[46] H. Sajjad, N. Durrani, F. Dalvi, Neuron-level Interpretation of Deep NLP

Models: A Survey, CoRR abs/2108.13138 (2021). URL: https://arxiv.org/abs/2108.13138. arXiv:2108.13138.

[47] A. Tamkin, T. Singh, D. Giovanardi, N. D. Goodman, Investigating transferability in pretrained language models, ArXiv abs/2004.14975 (2020).

[48] A. Merchant, E. Rahimtoroghi, E. Pavlick, I. Tenney, What happens to bert embeddings during fine-tuning?, ArXiv abs/2004.14448 (2020).

[49] N. Durrani, H. Sajjad, F. Dalvi, How transfer learning impacts linguistic knowledge in deep NLP models?, in: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, Association for Computational Linguistics, Online, 2021, pp. 4947–4957. URL: https://aclanthology.org/2021.findings-acl.438. doi:10.18653/v1/2021.findings-acl.438.

[50] F. Dalvi, N. Durrani, H. Sajjad, Y. Belinkov, D. A. Bau, J. Glass, What is one grain of sand in the desert? analyzing individual neurons in deep nlp models, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI, Oral presentation), 2019.

[51] N. Durrani, H. Sajjad, F. Dalvi, Y. Belinkov, Analyzing individual neurons in pre-trained language models, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Online, 2020, pp. 4865–4880. URL: https://aclanthology.org/2020.emnlp-main.395. doi:10.18653/v1/2020.emnlp-main.395.

[52] T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, Y. Artzi, Revisiting few-sample bert fine-tuning, 2020. arXiv:2006.05987.

[53] F. Dalvi, A. R. Khan, F. Alam, N. Durrani, J. Xu, H. Sajjad, Discovering latent concepts learned in BERT, in: International Conference on Learning Representations, 2022. URL: https://openreview.net/forum?id=POTMtpYI1xH.

[54] H. Sajjad, N. Durrani, F. Dalvi, F. Alam, A. R. Khan, J. Xu, Analyzing encoded concepts in transformer language models, in: Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '22, Association for Computational Linguistics, Seattle, Washington, USA, 2022.

[55] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Seattle, Washington, USA, 2013, pp. 1631–1642. URL: https://www.aclweb.org/anthology/D13-1170.

[56] A. Williams, N. Nangia, S. Bowman, A broad-coverage challenge corpus for sentence understanding through inference, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana, 2018, pp. 1112–1122. URL: https://www.aclweb.org/anthology/N18-1101. doi:10.18653/v1/N18-1101.

[57] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang, SQuAD: 100,000+ questions for machine comprehension of text, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Austin, Texas, 2016, pp. 2383–2392. URL: https://www.aclweb.org/anthology/D16-1264. doi:10.18653/v1/D16-1264.

[58] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, B. Magnini, The fifth pascal recognizing textual entailment challenge, in: In Proc Text Analysis Conference (TAC'09, 2009.

[59] W. B. Dolan, C. Brockett, Automatically constructing a corpus of sentential paraphrases, in: Proceedings of the Third International Work-

shop on Paraphrasing (IWP2005), 2005. URL: https://www.aclweb.org/anthology/I05-5002.

[60] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, L. Specia, SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation, in: Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), Association for Computational Linguistics, Vancouver, Canada, 2017, pp. 1–14. URL: https://www.aclweb.org/anthology/S17-2001. doi:10.18653/v1/S17-2001.

[61] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Brew, Huggingface's transformers: State-of-the-art natural language processing, ArXiv abs/1910.03771 (2019).