EFFICIENT STAGEWISE PRETRAINING VIA PROGRESSIVE SUBNETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent developments in language models have sparked interest in developing efficient pretraining methods. A recent and effective paradigm is to perform stagewise training, where the depth of the model is gradually increased over the course of training starting from a shallow network (e.g. gradual stacking (Reddi et al., 2023)). While this is appealing since it yields resource and wall-time savings, it has limitations, particularly the inability to assess and evaluate the full model performance during earlier stages, and degradation in model quality due to smaller capacity of models in the initial stages. In this work, we propose an alternative framework, *progressive subnetwork training*, that maintains the full model throughout training, but only trains subnetworks within the model in each step. We empirically focus on a simple instantiation of this framework — **Ra**ndom **P**ath **Tr**aining (RAPTR) — that only trains a sub-path of layers in each step, progressively increasing the path lengths in stages. We demonstrate that RAPTR achieves better pre-training loss for BERT and UL2 language models while requiring 20-33% fewer FLOPs compared to standard training, and is competitive or better than gradual stacking at similar FLOPs. Furthermore, RAPTR shows better downstream performance on UL2, improving multiple QA and SuperGLUE tasks by 1-5% compared to standard training and stacking. Finally, we provide theoretical basis of RAPTR for residual networks by characterizing their *stability* due to residual connections and layer norm.

1 INTRODUCTION

Large network based language models (e.g. Transformers) have revolutionized the field of NLP. Intriguingly, these language models have demonstrated remarkable *emergent abilities* that only begin to manifest at large scales. However, training such large models is usually very slow and resource intensive (Brown et al., 2020; Touvron et al., 2023; Chowdhery et al., 2022). This has sparked interest in efficient training of large models, necessitating the development of new algorithms and paradigms for efficient pretraining. Traditionally, this was accomplished by designing better optimization algorithms (e.g. (Liu et al., 2023a; Chen et al., 2023; Gupta et al., 2018; Shazeer & Stern, 2018)) that require fewer training steps to reduce the loss. Recently, other training paradigms have garnered interest. The most prominent in this line of research is that of *stagewise training*. These approaches typically train large models in stages, starting with cheaper training in the early stages, which can find a good initializer for expensive later stages. This powerful framework was shown to drastically reduce the training iterations need to train the full large model.

Gradual stacking (Reddi et al., 2023) and Progressive stacking (Gong et al., 2019) are instantiations of aforementioned stagewise training paradigm. Progressive stacking trains deep neural networks in stages, starting with a small network and doubling the number of layers at each stage, until the desired depth is reached. At each stage, the parameters of the previous stage are stacked onto itself to initialize the larger network. Instead of simply doubling the network, Gradual stacking proposes to increase the network depth in a gradual manner; thereby, yielding better resources and wall-time savings. Overall, by using this framework, Reddi et al. (2023) demonstrated that one can obtain more than 2x speed up in BERT pretraining by gradually growing the network depth. They also empirically demonstrate the training performance is sensitive to the (stacking) schedules; thus, the optimal schedule indeed requires a careful grid search. Furthermore, it is not possible to assess and evaluate the performance of the full model during earlier stages of stacking (i.e., it is not an anytime

algorithm). It should also be emphasized that stacking approaches use a much smaller model for part of the training, which can possibly hamper the quality of the model (e.g. long term memory of the Transformer (Geva et al., 2021)). Intuitively, this is especially problematic for single epoch training of language models where each data sample is only used once during training. As we shall see later, this is reflected in the downstream performance of stacking based approaches (see section 3.2).

In this paper, we study a novel stagewise training approach which addresses the above limitations. Unlike stacking, where different stages train models of different sizes, we maintain a common base model of interest and train sub-networks of *varying complexities* within this model. More precisely, in each stage we perform forward and backward passes over a randomly sampled subset of layers within the model, progressive increasing the number of layers in each stage. This can be achieved using a simple recipe: (a) drop out layers randomly with certain probability during forward and backward passes (along with careful rescaling), and (b) progressively *decrease* the dropout probability to zero over time. Similar to stacking, this can reduce the total FLOPs in the initial stages while providing a good initialization for the final full model training. However, in contrast to stacking, it allows us to track the full model throughout training. We demonstrate that this simple strategy is very effective at making training more efficient.

While dropping layers during training has been explored in the past, their motivation is either efficient inference by reducing effective model depth, regularization effects or saving wall-clock towards the end of training by increasingly dropping more layers later (Fan et al., 2019; Zhang et al., 2019; Liu et al., 2023c). Our idea of progressive subnetwork training is fundamentally different in motivation and implementation from these works (see Section 5). We now state the following primary contributions of this paper.

- We introduce a novel stagewise training subnetwork (section 2) that enables tracking the performance of the final model in all stages. Specifically we explore Random Path training (RAPTR) that trains a path comprised of a random subset of layers, with the average length of path progressively increasing in stages. We introduce ideas like scaling outputs for missing layers and fixing some layers in the net for improved performance.
- We conduct extensive comparisons between gradual stacking and RAPTR on BERT (Devlin et al., 2018) and UL2 language models. On BERT-Base, RAPTR demonstrates notable improvements in the pretraining loss when compared to gradual stacking at similar FLOPs, while also being better than baseline training with 33% lesser FLOPs. The benefits of RAPTR on BERT training are especially significant in the short horizon setting (few epoch training).
- For UL2-$1.6B$ language model (Tay et al., 2022), RAPTR can match the pretraining perplexity of baseline training with 20% fewer FLOPs. Interestingly, we observe significant improvements in downstream evaluations, e.g. we see 3-5% absolute increase in 1-shot exact match scores on TydiQA (Clark et al., 2020) and SQuADv2 (Rajpurkar et al., 2018) compared to baseline and gradual stacking. Additionally, compared to baseline training, RAPTR has a lot lesser variance over training runs, indicating implicit benefits of stage-wise training for downstream performance stability (section 3.2).
- We provide a theoretical study to analyze sufficient conditions under which RAPTR yields good initializers across stage transitions (section 4). We characterize a stability condition with respect to dropping of layers that can lead to smooth transition of loss across stages. Illustrative examples with linear networks are presented to show how layernorm and residual connections can help with this stability.

2 PROGRESSIVE SUBNETWORK TRAINING

Notation. We use $[n]$ denotes the set $\{1, 2, \ldots, n\}$. $a_{1:k}$ is used to denote a sequence of k scalars $a_1, \ldots, a_k \in \mathbb{R}$, and $\boldsymbol{x}_{1:k}$ to denote a sequence of k vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^d$. We alternately also use $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ to denote a matrix containing a sequence $\boldsymbol{x}_{1:n}$ as rows.

We consider the setting where the goal is to learn an L-layer sequence-to-sequence neural network. (e.g. standard Transformer network).

Definition 2.1 (L-layer sequence-to-sequence model)**.** Let $f_1, \cdots, f_L : \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d}$ denote a set of L sequence-to-sequence functions Then, $F : \mathbb{R}^{N \times d} \times \mathbb{R}^L \to \mathbb{R}^{N \times d}$ with a sequence

input $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and L scalars $\alpha_{1:L}$, outputs a sequence $\boldsymbol{Y}$ after a recursive computation with intermediate sequences $\boldsymbol{Y}^{(1)}, \cdots, \boldsymbol{Y}^{(L-1)}, \boldsymbol{Y}^{(L)}$, given by

$$\boldsymbol{Y}^{(\ell)} = \boldsymbol{Y}^{(\ell-1)} + \alpha_j f_j(\boldsymbol{Y}^{(\ell-1)}) \text{ for all } 1 \leq \ell \leq L.$$

Here $\boldsymbol{Y}^{(0)}$ denotes $\boldsymbol{X}$ for simplicity and output $\boldsymbol{Y}$ is equal to $\boldsymbol{Y}^{(L)}$.

Standard model training is represented by the backbone function F, where $\alpha_i = 1$ for $i \in [L]$. For simplicity, given an input sequence $\boldsymbol{X}$, we will use $F(\boldsymbol{X})$ as the output and $\boldsymbol{Y}^{(1)}, \boldsymbol{Y}^{(2)}, \cdots, \boldsymbol{Y}^{(L)}$ as the intermediate layer outputs of F under standard model training, unless specified otherwise. The output of the model is passed through a function $H : \mathbb{R}^d \to \mathbb{R}^V$ that projects the model output into V-dimensional logits, where V is the vocabulary size. For an example $\boldsymbol{X}$, cross-entropy loss based on true labels is denoted as $\mathcal{L}(H \circ F(\boldsymbol{X}))$ (or $\mathcal{L}(F(\boldsymbol{X}))$ when H is clear from context). Note that this notation hides the true labels for the ease of exposition. The final loss is then defined as $\mathcal{L}(F) = \mathbb{E}_{\boldsymbol{X} \sim \mathcal{D}}[\mathcal{L}(F(\boldsymbol{X}))]$ for the input distribution $\mathcal{D}$.

Given such a setting, *progressive subnetwork training*, a stagewise training framework, consists of two crucial components:

1. **Subnetwork selection & rescaling:** At each iteration, we select a (possibly random) subnetwork of the L-layer neural network. The forward and backward passes are computed based on this subnetwork network along with appropriate scaling (e.g. Section 2.1)).
2. **Progressive subnetworks:** In stages, we progressively increase the size of the subnetworks starting from small subnetworks and end with the complete network in the final stage.

We note that this framework is quite general. To make it more concrete, in the following section, we provide a simple but powerful instantiation of the framework based on *random paths*.

2.1 Random Path Training: RaPTr

One simple approach to select a subnetwork is by skipping a few layers and choosing a *path*. More concretely, given a network F to be trained, we pick a random subset of layers for the forward pass and bypassing the rest of the layers using the residual connections. Let p denotes the probability of randomly selecting a layer subnetwork and $\mathcal{I}$ denotes a subset of layers that are always included, i.e. they are never bypassed. We define the following before describing RaPTr in more detail.

Definition 2.2 ($(p, \mathcal{I})$-subnetwork)**.** Let $\zeta_{1:L}$ be Bernoulli samples, with $\zeta_i = 1$ for all $i \in \mathcal{I}$, while $\zeta_i \sim B(p)$ for all $i \notin \mathcal{I}$. The set of all layers for which $\zeta_i = 1$ represents a random subnetwork while layers with $\zeta_i = 0$ are bypassed. The output of the selected subnetwork on a sequence input $\boldsymbol{X}$ is equivalent to the output of the model given by $F(\boldsymbol{X}, \zeta_{1:L})$.

The pseudo-code for RaPTr is provided in Algorithm 1. On a high level, the total training is split into k stages. Each stage s uses $(p_s, \mathcal{I}_s)$-subnetwork between steps T_s and T_{s+1}. We denote this schedule by $T_{1:k} \times (p_{1:k}, \mathcal{I}_{1:k})$. Importantly, we progressively increase the random subnetwork selection pattern across stages i.e., the expected length of the selected random subnetworks is increased progressively towards full model training. This can be achieved by either increasing the probability of including each layer in random subnetwork or fixing more layers in $\mathcal{I}$. or both. More formally, we use schedules $T_{1:k} \times (p_{1:k}, \mathcal{I}_{1:k})$ that satisfies: (a) $p_s \leq p_{\tilde{s}}$, and (b) $\mathcal{I}_s \subseteq \mathcal{I}_{\tilde{s}}$ for all $1 \leq s \leq \tilde{s} \leq k$.

Note that each training step in RaPTr involves computing forward and backward passes only on random subnetwork. In expectation, training a $(p, \mathcal{I})$ random subnetwork will require $\frac{|\mathcal{I}| + (L - |\mathcal{I}|)p}{L}$ FLOPs relative to standard training that uses all L layers. By using $p_s \ll 1$ for most part of training, RaPTr can significantly increase the training efficiency.

Scaling to maintain identity. While working on subnetworks, it is important to appropriately rescale the network. In particular, bypassing layer(s) in RaPTr can shift the input distribution for layers compared to the full model. To mitigate this shift, we scale the output of the layers to maintain the norms of the input to each layer throughout training. The idea of scaling layer outputs has also been explored in prior work Huang et al. (2016); Fan et al. (2019).We use a different square-root scaling mechanism that is motivated by analysis on a randomly initialized Transformer architecture. We show that at initialization, the norm of the intermediate outputs $\boldsymbol{y}^{(\ell)}$ scale as $\sqrt{\ell}$.

Algorithm 1: k-stage RAPTR

Require: Blocks $f_{\theta_1}, \cdots, f_{\theta_L}$, Schedule $T_{1:k} \times (p_{1:k}, \mathcal{I}_{1:k})$, total steps T, data and label distribution D. Boolean $scale$ whether to scale unskipped layers.
Initialize $\theta_1, \cdots, \theta_L$, $T_{k+1} = T$.
for $s = 1 \to k$ **do**
  for $t = T_s \to T_{s+1}$ **do**
    Sample batch $(\boldsymbol{X}, \boldsymbol{Y}) \sim D$.
    Sample $\zeta_i \sim B(1)$ for all $i \in \mathcal{I}_s$, and $\zeta_i \sim B(p)$ for all $i \notin \mathcal{I}_s$.
    if *scale is True* **then** $\overline{\zeta}_{1:L} = h_{sqrt}(\zeta_{1:L})$;
    else $\overline{\zeta}_{1:L} = \zeta_{1:L}$;
    Set $F_0(\boldsymbol{X}, \overline{\zeta}_{1:L}) = \boldsymbol{X}$
    for $j = 1 \to L$ **do**
      if $\overline{\zeta_i} = 0$ **then** $F_j(\boldsymbol{X}, \overline{\zeta}_{1:L}) = F_{j-1}(\boldsymbol{X}, \overline{\zeta}_{1:L})$;
      else $F_j(\boldsymbol{X}, \overline{\zeta}_{1:L}) = \overline{\zeta}_j f_{\theta_j}(F_{j-1}(\boldsymbol{X}, \overline{\zeta}_{1:L})) + F_{j-1}(\boldsymbol{X}, \overline{\zeta}_{1:L})$;
    end for
    Compute loss with output $F_L(\boldsymbol{X}, \overline{\zeta}_{1:L})$ and labels $\boldsymbol{Y}$, and take a gradient step.
  end for
end for

Theorem 2.3 (Informal, cf theorem F.11)**.** *At random initialization, given any input sequence $\boldsymbol{x}_{1:N}$ and scalars $\alpha_{1:L}$, with high probability the intermediate layer outputs of F (definition 2.1) satisfy*

$$\left\| \boldsymbol{y}_i^{(\ell)} \right\|_2^2 = \|\boldsymbol{x}_i\|_2^2 + \Theta(\sum_{j=1}^{\ell} \alpha_j^2 d), \quad \textit{for all } 1 \le i \le N, 1 \le \ell \le L.$$

Inspired by this, we define the function h_{sqrt} that takes in a sequence of binary values $\zeta_{1:L} \in \{0,1\}^L$ and returns scalars $\overline{\zeta}_{1:L}$ where $\overline{\zeta}_i$ is the scaling for output of layer i. We find a scaling that satisfies the following two conditions hold (a) $\sum_{i=1}^{j} \overline{\zeta_i}^2 = j$ for all $j \le L$ (maintaining the norm), and (b) $\overline{\zeta_i} = 0$ if $\zeta_i = 0$ for all $i \le L$ (maintaining layers in random subnetwork). Formally, for each index j, it finds the index minimum $\bar{j} > j$ with $\zeta_{\bar{j}} = 1$ and sets $\overline{\zeta}_j = \sqrt{\bar{j} - j}$. The RAPTR algorithm with square root scaling is presented in algorithm 1. In Figure 1 we compare square root scaling with some other scaling methods and find it to be slightly better.

Fixing first and last layers. For RAPTR, we chose to always keep the first and layers during training. The intuition is that since the first and last layers play different roles compared to the middle layer because they touch the input and output embeddings. Ablations in Figure 1 suggest that fixing these layers improves the pretraining loss throughout the trajectory. In particular, for an L-layer network we fix the set $\mathcal{I} = \{1, L\}$ (Definition 2.2) for most subsequent experiments.

Next, we discuss our empirical findings before proceeding to the theoretical analysis (Section 4).

3 EXPERIMENTS

In this section, we present comprehensive experiments that compare RAPTR with standard baseline training and gradual stacking for standard BERT and UL2 language models. We primarily choose stacking for comparison given the recent findings in (Kaddour et al., 2023) that show stacking to be the best (and only) efficient training method that can beat baseline over different training horizons.

Notations for RAPTR and stacking schedules. We succinctly denote the stage schedules by a set of network lengths separated by hyphens. For gradual stacking, schedule 6-12-18-24 refers to 4 stages with a 6, 12, 18 and 24 layer model training respectively. For RAPTR training with a 24-layer model, 6-12-18-24 refers to 4 stages, with $(p, \mathcal{I}) = (\frac{4.}{22.}, \{1, 24\})$ random subnetwork training in the first stage which translates to sub-nets of length 6, followed by $(\frac{10.}{22.}, \{1, 24\}), (\frac{16.}{22.}, \{1, 24\}), (1., \{1, 24\})$ in the next 3 stages. These correspond to $12, 18$, and 24 expected length subnetwork respectively. Unless specified otherwise, we maintain an equal number of training steps for each stage.

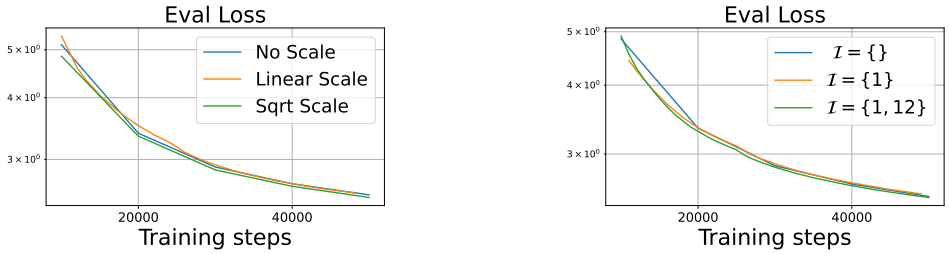


Figure 1: Ablation study on choices for RAPTR algorithm. We train BERT-base with RAPTR for 100k steps with a 6-8-10-12 schedule (see Section 3). Left to right: (a) Square-root scaling (h_{sqrt}) has better validation loss compared with linear scaling and no scaling at training, especially in the earlier stages. With *no scale*, we scale layer's output by $1/p_i$ during inference (following Huang et al. (2016)). (b) Different candidates for the fixed set $\mathcal{I}$ are compared for RAPTR. We find that training with first and last layers fixed helps faster training.

3.1 EXPERIMENTS WITH BERT

Experiment setup. We pretrain BERT models (Devlin et al., 2018) on Wikipedia + Books dataset with AdamW optimizer (Loshchilov & Hutter, 2019). For all the experiments with BERT-BASE and BERT-LARGE, the total number of training steps is set to 675K and 1M steps, respectively. We report the evaluation loss and fine-tuning performance of the trained models on 3 GLUE tasks (Wang et al., 2018). Please see appendix B for more experimental details.

Results at equal training steps. We observe that RAPTR achieves similar or better evaluation loss to baseline training, despite baseline training using 33% more FLOPs than RAPTR for BERT-BASE. Compared to gradual stacking at similar FLOPs, RAPTR again has better validation loss. Additionally, all three methods exhibit similar performance in downstream fine-tuning tasks (see Table 1).

Results at equal FLOPs. Inspired by Kaddour et al. (2023), we further compare RAPTR and gradual stacking to baseline training by adjusting the number of training steps of baseline training to match its FLOPs to RAPTR. In Table 2, we compare these methods at three different FLOPs budgets. For BERT-BASE, we observe that at shorter FLOP experiments, RAPTR achieves better much validation loss compared to baseline training and gradual stacking. This difference gets smaller as we move to larger horizon settings. Similar findings for BERT-LARGE.

Table 1: Loss and fine-tuning results on 12-layer BERT at 675K steps and equal peak learning rate (10^{-4}). Key observations: (a) RAPTR achieves similar loss as baseline at 75% reduced FLOPs. (b) RAPTR achieves 0.02 better evaluation loss than gradual stacking for same schedules. (c) All methods exhibit similar performance in downstream fine-tuning task, with RAPTR having a slight edge.

	Relative FLOPs	Eval Loss	MNLI	QNLI	SST-2
Baseline	1.33	1.76	81.5	**90.5**	91.4
Gradual Stacking					
6-8-10-12	1	1.78	-	-	-
6-9-12	1	1.77	80.9	89.8	91.1
RAPTR					
6-8-10-12	1	**1.75**	**82.1**	89.8	**92.4**
6-9-12	1	1.75	-	-	-

3.2 EXPERIMENTS WITH UL2-1.6B

Experiments on BERT (Section 3.1) show that RAPTR and stacking outperform baseline training at equal FLOP comparisons for short horizon settings, equivalently with few training epochs. This

Table 2: Equal FLOPs comparisons for BERT-BASE and BERT-LARGE with extensive peak LR tuning. For BERT-BASE we use the schedule 6-8-10-12 for RAPTR and 6-9-12 for stacking. For BERT-LARGE, we use 6-12-18-24 for both RAPTR and stacking. FLOPs denotes the number of steps involved in baseline training. Key observations: for BERT-BASE (a) RAPTR achieves better loss compared to baseline at all FLOP measures, with larger differences for fewer FLOPs. (b) RAPTR has 0.02 better loss than gradual stacking at all FLOP levels. (c) For BERT-LARGE, stacking and RAPTR are competitive to each other at all FLOP counts. Both methods have $0.03 - 0.07$ better loss than baseline at lower FLOP measures.

BERT-base				BERT-large			
FLOPs	Baseline	Stacking	RAPTR	FLOPs	Baseline	Stacking	RAPTR
$75k$	2.09	2.02	**2.01**	$62.5k$	1.84	**1.78**	1.80
$170k$	1.90	1.88	**1.86**	$140k$	1.63	**1.60**	1.61
$510k$	1.74	1.75	**1.73**	$625k$	**1.40**	1.41	1.41

prompts us to consider whether similar disparities emerge in scenarios where only one or a few training epochs are feasible, e.g. training billion-parameter language models on large text corpora.

We pretrain a 1.6B decoder-only UL2 model (Tay et al., 2022) with 24 layers. We use Adafactor (Shazeer & Stern, 2018) optimizer and train with a batch size 512 for $400k$ steps on a mixture of Arxiv, C4 (Raffel et al., 2020), Github, and Wikipedia (Foundation) datasets, with mixing ratios $9\%, 57\%, 17\%, 17\%$ respectively. This roughly corresponds to 0.8 epochs of C4. Table 3 reports the validation loss and downstream 1-shot performance without fine-tuning. Reported datasets include Trivia QA (Joshi et al., 2017), Tydi QA (Clark et al., 2020), SQuADV2 (Rajpurkar et al., 2018), and SuperGLUE (Wang et al., 2019). For QA tasks, we report Exact match scores, and report average accuracy for SupreGLUE tasks. To reduce variance, we report average performance across 3 runs for the most representative setting for each method. Please see appendix B for more experimental details.

Schedules for RAPTR and gradual stacking. In table 3, we report for a schedule with 4 stages, denoted by 12-16-20-24. The length of each stage has been adjusted appropriately to use 17% lower FLOPs (equivalent to training with average path length of 20 out of 24) compared to the baseline training. We evaluate an alternative subnetwork configuration that involves an initial full-model training phase for 30K steps before transitioning to the RAPTR schedule. For parity in FLOPs, we reduce 30K steps from the final training phase.

Results. Firstly at 20% FLOPs benefit, we find in Table 3 that RAPTR with 30K steps initial full model training achieves similar evaluation perplexities to baseline training. Stacking on the other hand is slightly worse at the same FLOPs as RAPTR. We find a better schedule for stacking and report results in the Appendix (see Table 4). Overall RAPTR is still better than stacking and is more robust to the schedule selection.

We also find that including 30K steps of full-model training at the start helps with all metrics, despite not changing FLOPs. This is an added benefit of subnetwork training over stacking. Intriguingly, we find that despite having similar validation perplexity, RAPTR has much better downstream metrics compared to both baseline and stacking. The improvements are particularly large for TydiQA (3.8%) and SuperGLUE (0.3%). This perhaps suggests an that RAPTR has a desirable implicit bias, a phenomenon that has been recently studied for other pretraining settings (Saunshi et al., 2022; Liu et al., 2023b). Another notable observation is that RAPTR has lower variance on all metrics and is thus more stable to training runs ($\approx 2\%$ lower variance in both SquADv2 and TydiQA), again pointing to some implicit benefits of stagewise training. We believe these inductive bias benefits of RAPTR deserves further exploration.

Following Kaddour et al. (2023), we repeat experiments with baseline training with reduced training steps to match total FLOPs to RAPTR and gradual stacking. Furthermore, for fair comparison, we report the model with the best evaluation perplexity after baseline training with peak learning rates tuned over $\{1, 1.5, 2\} \times 10^{-2}$ grid. Despite the tuning, we find RAPTR to have much better perplexity and downstream metrics in this equal FLOPs setting.

Table 3: Validation loss and 1-shot downstream evals on UL2-1.6B. Key observations: (a) RAPTR with 30k initial training improves performance by atleast $2-4\%$ on Trivia QA and SquADv2. RAPTR is atleast 5% better than gradual stacking on various downstream tasks, (b) Compared to baseline, RAPTR is $1-2\%$ better on all downstream tasks on average. Reducing number of steps of the baseline to match FLOPs hurts the baseline's performance even after extensive tuning of peak learning rate. (c) Stage-wise training achieves $(1-2\%)$ lower variance on Tydi QA and SquADv2, implying implicit benefits of stagewise training. † We report the best performance after tuning with 3 learning rates $\{1, 1.5, 2\} \times 10^{-3}$

	Rel. FLOPs	Eval Loss	Trivia QA	Tydi QA	SQuADv2	SuperGLUE
Baseline	1.2	2.06 (0.01)	25.0 (0.2)	34.4 (3.1)	42.1 (2.9)	60.0 (0.4)
Equiflop Baseline†	1	2.07 (—)	24.3 (—)	35.7 (—)	42.37 (—)	60.6 (—)
12-16-20-24 Stacking	1	2.08 (0.00)	20.1 (1.3)	28.6 (2.4)	36.0 (1.9)	60.4 (0.9)
12-16-20-24 RAPTR	1	2.08 (—)	22.2 (—)	**38.2** (—)	40.6 (—)	60.1 (—)
(+30k initial full-model train)	1	**2.06** (0.00)	**25.8** (0.2)	36.7 (1.0)	**44.1** (0.5)	60.9 (0.2)

4 THEORETICAL FRAMEWORK USING LAYER STABILITY

This section provides insights and theoretical evidence for RAPTR approach. In our experiments, we observe that the pretraining loss changes somewhat smoothly while transitioning from one stage to another (please see fig. 4 in the appendix). This phenomenon suggests that the model trained with shorter paths in an earlier stage is a good initializer for the next stage that uses longer paths. Here we will examine the factors that help the model maintain its stability during the transitions between stages. Inspired by Arora et al. (2018), we hypothesize that the model output is relatively stable to noise introduced due to random subnetwork training . This stability to perturbations ensures that the pretraining loss of the model does not deviate much as we switch from shorter subpaths to longer ones in the next stage. The rest of the section formalizes this stability argument and shows how residual connection and layernorm can help. All norms in this section refer to ℓ_2 norm of a vector.

4.1 SUBNETWORK STABILITY

For simplicity, we restrict the discussion to sequences of length $N = 1$; generalization to $N > 1$ is fairly easy. To understand loss changes across stage boundaries, we consider two stage RAPTR: the first stage trains with subnetwork of length $L-1$ by dropping a single layer at random, and the second stage trains the entire model. Suppose $\mathcal{L}_1$ and $\mathcal{L}_2$ denote the effective loss functions being minimized in the two stages. The aim is to show that F learned in the first stage by minimizing $\mathcal{L}_1$ should also result in small value for $\mathcal{L}_2$.

Definition 4.1. Let $F_{-\ell}$ denote the subnetwork obtained by skipping layer ℓ. The stability of network output with respect to dropping layer ℓ is defined as $\Psi_\ell(\boldsymbol{x}) = \|F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x})\|_2$

Our claim, that is formalized in Theorem 4.2 is that $\mathcal{L}_2(F) - \mathcal{L}_1(F)$ is small if the stability scales slower than the norm of the output.

Theorem 4.2 (Informal, cf theorem G.1)**.** *The difference between expected loss of* $(L-1)$-RAPTR *and* L-RAPTR*, i.e.* $|\mathcal{L}_2(F) - \mathcal{L}_1(F)|$*, is upper bounded by* $\frac{C}{L}\mathbb{E}_{\boldsymbol{x}\in D}(\sum_{\ell=1}^{L}\Psi_\ell(\boldsymbol{x}))/\|F(\boldsymbol{x})\|_2$ *for some constant* C *that depends on regularity properties of the head* H *on top of the backbone* F*.*

The proof follows by observing that $\mathcal{L}_1(F) = \frac{1}{L}\sum_{\ell\in[L]}\mathcal{L}(F_{-\ell})$ and $\mathcal{L}_2(F) = \mathcal{L}(F)$. Thus, the losses are close if $F_{-\ell}$ is close enough to F for a random ℓ. This result shows that difference in loss functions between two stages is small if the stability of the network output is small relative to the network output. Thus, the relative stability of the network determines the success of the approach. For the purpose of illustration, below we analyze these quantities for the special case of linear residual networks.

4.2 ILLUSTRATIVE EXAMPLE: LINEAR NETWORKS

We consider a residual network where the layers $f_{1:L}$ are linear with parameters $\boldsymbol{W}_{1:L}$. The layer output is $\boldsymbol{y}^\ell = \boldsymbol{y}^{(\ell-1)} + \boldsymbol{W}_\ell \boldsymbol{y}^{(\ell-1)}$ or $\boldsymbol{y}^\ell = \boldsymbol{y}^{(\ell-1)} + \boldsymbol{W}_\ell \boldsymbol{y}^{(\ell-1)}/\left\|\boldsymbol{y}^{(\ell-1)}\right\|_2$ depending on whether

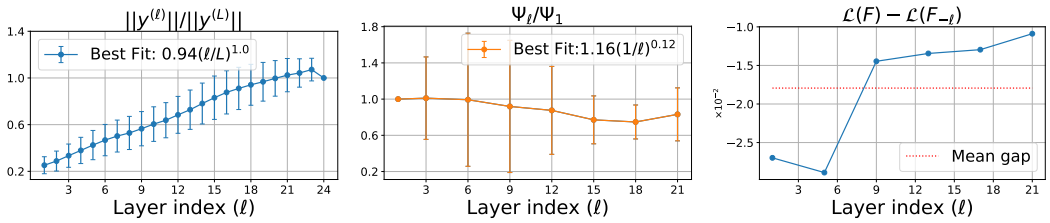


Figure 2: Stability study on BERT-LARGE trained for $50k$ steps with $(L-1)$ RAPTR. Left to right: Behavior of (a) norms of intermediate activations $\boldsymbol{y}^{(\ell)}$, (b) Ψ_ℓ/Ψ_1 (definition 4.1), and (c) Loss gap between different random subnetwork $F_{-\ell}$ and model F, given by $\mathcal{L}(F) - \frac{1}{L}\sum_{\ell=1}^{L}\mathcal{L}(F_{-\ell})$. Key observations: (a) Norms of the intermediate activations grow linearly with ℓ, (b) Ψ_ℓ changes slowly with ℓ as $(\frac{L}{\ell})^{0.12}$, suggesting a worse-case bound of $\mathcal{O}(L^{-0.88})$ on $\mathcal{L}(F) - \frac{1}{L}\sum_{\ell=1}^{L}\mathcal{L}(F_{-\ell})$ based on Theorem 4.2 (c) Interestingly, $\mathcal{L}(F) \leq \frac{1}{L}\sum_{\ell=1}^{L}\mathcal{L}(F_{-\ell})$, even when model is trained with $L-1$ random subnetworks. We leave this observation for careful study in the future.

layernorm is enabled or disabled respectively . We also consider another setting which has layernorm but no residual connection; so $\boldsymbol{y}^{\ell} = \boldsymbol{W}_\ell \boldsymbol{y}^{(\ell-1)}/\left\|\boldsymbol{y}^{(\ell-1)}\right\|_2$. The following result studies the scale of Ψ_ℓ and output norm from Theorem 4.2 for these setting.

Lemma 4.3. *At random initialization where $\boldsymbol{W}_i \sim \mathcal{N}(0, d^{-1/2}\boldsymbol{I})$, for any input $\boldsymbol{x} \in \mathbb{S}^{d-1}$, we have*

(a) With residual connection & layernorm, we have $\Psi_\ell(\boldsymbol{x}) = \mathcal{O}(\sqrt{L/\ell})$ and $\|F(\boldsymbol{x})\|_2 = \Omega(\sqrt{L})$. Thus $\frac{1}{L}\sum_{\ell=1}^{L}\Psi_\ell(\boldsymbol{x}) = \mathcal{O}(1)$ and the gap in losses between stages is $\mathcal{O}(1/\sqrt{L})$.

(b) Without residual connection, we have both $\Psi_\ell(\boldsymbol{x}) = \Omega(1)$ and $\|F(\boldsymbol{x})\|_2 = \mathcal{O}(1)$. Thus the gap in losses between stages is $\Omega(1)$.

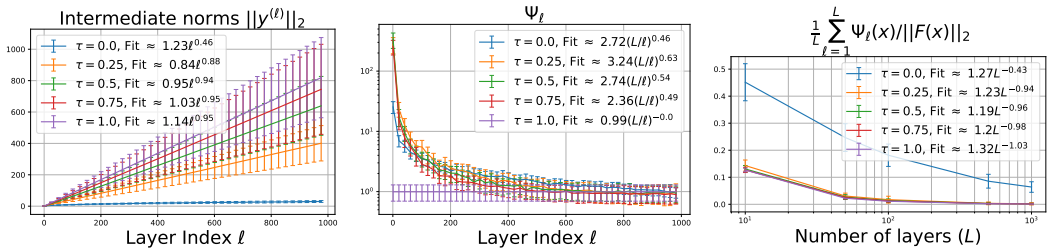
(c) Without layernorm, we have $\Psi_\ell(\boldsymbol{x}) = \Omega(2^{(L-1)/2})$ and $\|F(\boldsymbol{x})\|_2 = \mathcal{O}(2^{L/2})$. Thus the gap in losses between stages is $\Omega(1)$.

In the appendix, we further show similar results for perfectly aligned layers, where the parameters of each layer are equal to some matrix $\boldsymbol{A}$ (lemma G.2). These examples captures the cases of perfectly correlated and uncorrelated layers and show stability to subnetwork training in the presence of input normalization layers. We can consider even more general scenarios, where the parameters of each layer are expressed as the combination of Gaussian matrices and a shared matrix (Figure 3). For a $\tau \in (0,1)$, the parameters of each layer ℓ is represented as $\sqrt{\tau}\boldsymbol{A} + \sqrt{1-\tau}\boldsymbol{G}^{(\ell)}$ for a shared matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ with $\|\boldsymbol{A}\|_2 \leq 1$ and $\boldsymbol{G}^{(\ell)} \sim \mathcal{N}\left(0, \frac{1}{\sqrt{d}}\boldsymbol{I}\right)$. We run simulations and experimentally observe that for each τ, $\Psi_\ell \approx (L/\ell)^{0.5}$ (Figure 3b), while the output norm grows faster as $\Omega(L^{0.88})$ (Figure 3a). A simple calculation shows that the gap between a $L-1$ random subnetwork and the full model scales as $\mathcal{O}(L^{-c})$ for some $c \geq 0.5$ (Figure 3c).

5 RELATED WORKS

The literature on training deep networks is vast; thus, we mainly focus on the most relevant works.

Stagewise efficient pretraining. The most relevant stagewise pretraining is stacking, which was introduced by Chen et al. (2015) for convolutional networks where they use function-preserving initialization from small models to train a deep neural network efficiently. Gong et al. (2019) modified this process for a transformer model, where they stack a shallow transformer to obtain a deeper model. This process has been further extended to other dimensions (e.g. width) by Gu et al. (2020), Shen et al. (2022), Wang et al. (2023), and Gesmundo & Maile (2023) in the form of growth operators ensuring the loss is preserved while growing the network. In this paper, we mainly focused on model depth expansion, but other growth operators also have an analog in our subnetwork training framework. This exploration is left for future work.



(a) (b) (c)

Figure 3: Behavior on a linear residual network with normalization layers with 100 random samples from $\mathbb{S}^{d-1}$, and dimension $d = 100$. The parameters of each layer ℓ is represented as $\sqrt{\tau}\boldsymbol{A} + \sqrt{1-\tau}\boldsymbol{G}^{(\ell)}$ for a shared matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ with $\|\boldsymbol{A}\|_2 \leq 1$ and $\boldsymbol{G}^{(\ell)} \sim \mathcal{N}\left(0, d^{-1/2}\boldsymbol{I}\right)$. Left to right: Behavior of (a) the norms of intermediate activation $\boldsymbol{y}^{(\ell)}$ with index ℓ, (b) Ψ_ℓ (definition 4.1) for each stack of layers $F_{\ell:L}$, and (c) $\frac{1}{L}\sum_{\ell=1}^{L} \Psi_\ell(\boldsymbol{x})/\left\|F(\boldsymbol{x})\right\|_2$ that appears in our bounds in theorem 4.2.

Stochastic depth. Our work is closely related to stochastic depth — a variant of structured dropout that drops layers at random with fixed probability during training. Stochastic depth is mainly motivated from the perspective of training extremely deep residual networks without increasing the training cost (Huang et al., 2016) . A key distinction is that these works keep the probability of dropping layers fixed during training; thus, stochastic depth, similar to dropout, can be viewed as a regularization mechanism, which is different from the role dropout plays in our work. Pham et al. (2019) train very deep Transformers for speech with this approach and show its regularization effect. Similar techniques have been used for vision transformers (Steiner et al., 2021; Tolstikhin et al., 2021). Fan et al. (2019) use fixed stochastic depth to train BERT models, and show that they can select subnetworks of any depth from one large network without having to fine-tune and with limited impact on performance, thus targeting inference efficiency as opposed to training efficiency.

Zhang & He (2020) used a schedule of stochastic depth for BERT pretraining where they explicity *increase* the drop rate of the layers towards the end of training. Kaddour et al. (2023) report that such a schedule, in fact, hurts the model performance significantly when compared to baseline and stacking. In striking contrast, our work shows that *decreasing* drop out schedule and careful rescaling enables faster training, and shows benefits in both pretraining and downstream performance (e.g. Section 3.2). Liu et al. (2023c) took a different approach by studying the regularization benefits of dropout and stochastic depth in training vision transformers to improves the test accuracy.

Subnetwork training. Training of random paths and subnetworks has been used in other contexts, e.g. parameter efficient fine-tuning (Houlsby et al., 2019; Pfeiffer et al., 2021; Hu et al., 2022; Liu et al., 2022) to reduce memory footprint, distributed and federated training (Dun et al., 2022) using multiple machines, and incremental learning (Jathushan et al., 2019) to avoid forgetting in continual learning. These ideas are not motivated by or used for reducing FLOPs during pretraining, to the best of our knowledge.

6 Conclusion and Future Work

This work proposes a stagewise training framework of *progressive subnetwork training* for efficient pretraining, and evaluates a natural instantiation of this framework (RaPTr) based on training random paths/subsets of layers. Overall RaPTr yields better quality language models than baseline training, while reducing the total FLOPs by at least 20-30%. At the same speedup, its also better the state-of-the-art depth-wise stacking approach. There are other stacking approaches with different growth functions in the literature (see Section 5 for references), and exploring analogs of subnetwork training for these growth functions could be a fruitful direction. From theory perspective, while the current analysis provides insights into why the loss does not deviate much at stage boundaries, it does not explain the intriguing stronger empirical finding that the loss *decreases* at the stage transitions. Additionally, there is evidence of certain desirable inductive biases of RaPTr style training, e.g. better downstream performance, smaller variance, which are not explained by our analysis. A better understanding of these phenomena may provide valuable insights into the relationship between various layers in the network, and could inspire other efficient training algorithms.

REFERENCES

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pp. 242–252. PMLR, 2019.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, pp. 254–263. PMLR, 2018.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pp. 1352–1361. PMLR, 2021.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544, 2013.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Shuning Chang, Pichao Wang, Hao Luo, Fan Wang, and Mike Zheng Shou. Revisiting vision transformer from the view of path ensemble. *arXiv preprint arXiv:2308.06548*, 2023.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, et al. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*, 2018.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

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

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*, pp. 2793–2803. PMLR, 2021.

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

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

Chen Dun, Cameron R Wolfe, Christopher M Jermaine, and Anastasios Kyrillidis. Resist: Layer-wise decomposition of resnets for distributed training. In *Uncertainty in Artificial Intelligence*, pp. 610–620. PMLR, 2022.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Wikimedia Foundation. Wikimedia downloads. URL https://dumps.wikimedia.org.

Andrea Gesmundo and Kaitlin Maile. Composable function-preserving expansions for transformer architectures. *arXiv preprint arXiv:2308.06103*, 2023.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pp. 2337–2346. PMLR, 2019.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the transformer growth for progressive bert training. *arXiv preprint arXiv:2010.12562*, 2020.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

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

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 646–661. Springer, 2016.

Rajasegaran Jathushan, Hayat Munawar, H Salman, Khan Fahad Shahbaz, and Shao Ling. Random path selection for incremental learning. *arXiv preprint*, 2019.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Jean Kaddour, Oscar Key, Piotr Nawrot, Pasquale Minervini, and Matt J Kusner. No train no gain: Revisiting efficient training algorithms for transformer-based language models. *arXiv preprint arXiv:2307.06440*, 2023.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Y Zhao, Yuexin Wu, Bo Li, et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. *arXiv preprint arXiv:2304.04947*, 2023.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023a.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning*, pp. 22188–22214. PMLR, 2023b.

Zhuang Liu, Zhiqiu Xu, Joseph Jin, Zhiqiang Shen, and Trevor Darrell. Dropout reduces underfitting. *arXiv preprint arXiv:2303.01500*, 2023c.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James Allen. Lsdsem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pp. 46–51, 2017.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39.

Ngoc-Quan Pham, Thai-Son Nguyen, Jan Niehues, Markus Müller, Sebastian Stüker, and Alexander Waibel. Very deep self-attention networks for end-to-end speech recognition. *arXiv preprint arXiv:1904.13377*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Sashank J Reddi, Sobhan Miryoosefi, Stefani Karp, Shankar Krishnan, Satyen Kale, Seungyeon Kim, and Sanjiv Kumar. Efficient training of language models using few-shot learning. 2023.

Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

Nikunj Saunshi, Jordan Ash, Surbhi Goel, Dipendra Misra, Cyril Zhang, Sanjeev Arora, Sham Kakade, and Akshay Krishnamurthy. Understanding contrastive learning requires incorporating inductive biases. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models. In *International Conference on Machine Learning*, pp. 19893–19908. PMLR, 2022.

Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, et al. Ul2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2022.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 32–42, 2021.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. *arXiv preprint arXiv:2303.00980*, 2023.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33:14011–14023, 2020.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

Table 4: Additional rows from table 3: we compare baseline runs at two learning rate schedules, and run compare RAPTR and Gradual stacking with a new 6-12-18-24 schedule. **Key Observations**: (a) Cosine decay learning works best for the baseline, compared to Square-root schedule proposed by Tay et al. (2022), (b) The performance of gradual stacking improves with the new schedule, implying a performance dependence with appropriate scheduling, and (c) RAPTR's performance doesn't change much with the new schedule.

	Rel. FLOPs	Eval Loss	Trivia QA	Tydi QA	SQuADv2	SGLUE
Baseline (Square-root LR decay)	1.2	2.07	23.4	31.9	44.3	60.0
Baseline (Cosine LR decay)	1.2	2.06	25.0	34.4	42.1	60.0
Equiflop Baseline (Square-root LR decay)	1	-	32.1	22.1	-	60.9
Equiflop Baseline (Cosine LR decay)	1	2.07	24.3	35.7	42.37	60.6
6-12-18-24 Stacking	1	2.06	22.1	34.6	38.0	60.5
12-16-20-24 Stacking	1	2.08	20.1	28.6	36.0	60.4
12-16-20-24 RAPTR	1	2.08	22.2	38.2	40.6	60.1
(+30k initial full-model train)	1	2.06	25.8	36.7	44.1	60.9
6-12-18-24 RAPTR (+30k initial full-model train)	1	2.06	24.2	37.3	42.3	61.1

A ADDITIONAL RELATED WORKS

Residual networks as ensembles. Training with random paths can be weakly viewed as ensembles over paths, and the connection between ResNets (He et al., 2016) and ensembles of shallow networks was first point out in Veit et al. (2016) for vision models. (Dong et al., 2021) showed the same phenomenon for self-attention models, where longer paths lead to rank-collapse in the self-attention module. Chang et al. (2023) study vision transformers (Dosovitskiy et al., 2020) as a cascade of multiple paths and propose pruning and self-distillation to remove long paths and improve performance. All these works mainly provide a novel perspective or inference efficiency but do not focus on training efficiency.

Learnable scaling of residual blocks. Bachlechner et al. (2021), Zhang et al. (2019), Touvron et al. (2021) consider learnable scales on the output of the residual blocks. These works aim to understand favorable initialization of the learnable scales under various constraints for faster training, which is very different from the role of scaling in our algorithm.

Early exit for efficient inference A lot of recent works have focused on improving inference efficiency for large language models. (Lei et al., 2023; Tay et al., 2022; Del Corro et al., 2023; Xin et al., 2020; Zhou et al., 2020; Hou et al., 2020). However, none of these works focus on efficient fine-tuning. Lei et al. (2023) modified pre-training by substituting resource-intensive MLP computations with straightforward classifiers for a predetermined fraction of tokens within the sequence. It's worth noting that their primary focus was on accelerating inference time rather than enhancing the efficiency of the pre-training process.

B ADDITIONAL EXPERIMENT DETAILS

B.1 DETAILS FOR BERT

Equal step Pretraining The batch-size is set to 256 and 512 for BERT-BASE and BERT-LARGE respectively. Following (Reddi et al., 2023), we use a learning rate warmup of 10k steps and keep learning rate constant in all phases, followed by a linear learning rate decay to 0 in the final full model training phase. We set the peak learning rate as 10^{-4} in all our experiments.

Equivalent flop Pretraining To make a fair comparison across all methods, we tune the peak the learning rate in the grid $\{1, 1.6, 2.5, 4\} \times 10^{-4}$ for BERT-BASE and grid $\{1, 1.6\} \times 10^{-4}$ for BERT-LARGE.

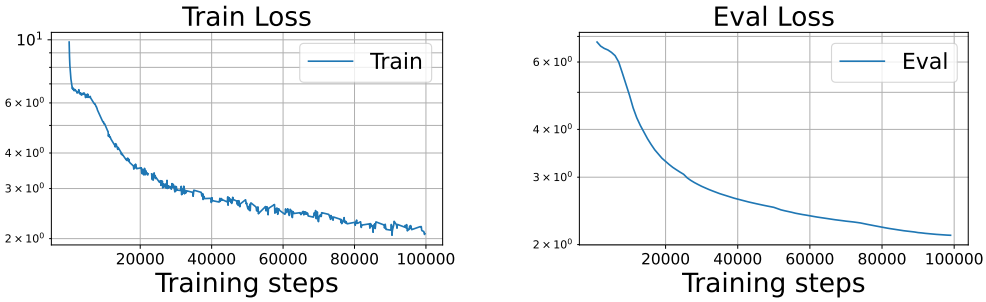


Figure 4: Train and Evaluation Loss behavior for a BERT-BASE model trained with RAPTR for 100k steps. We have 4 stages with 6-8-10-12 schedule (see section 3 for details). The boundaries are at 25k, 50k, and 75k. Key observation: the model's train and evaluation loss change smoothly across stage transitions, indicating stability of the model to subnetwork training.

Finetuning We fine-tune BERT models for 3 epochs with batch-size 32, using a learning rate grid search over $\{1, 2, 3, 5\} \times 10^{-5}$. We use a linear warmup schedule over initial 6% training steps, followed by constant learning rate schedule for the rest of the training.

B.2 DETAILS FOR UL2

Pretraining Similar to our experiments on BERT, we use a linear warmup of 10K steps, followed by constant learning rate in all stages, and a cosine decay to $0.1\times$ the peak learning rate in the final full model training stage. The peak learning rate is set to 10^{-2}.

Boundaries for 12-16-20-24 RAPTR and Gradual Stacking are 40k, 120k, and 240k. The average subnetwork length used during training turns out to be 20 with this schedule. When we introduce an initial 30k full-model training for RAPTR, we shift the boundaries by 30k so that the average subnetwork length stays 20.

Additional Pretraining experiments In table 4, we have the following additional runs compared to table 3 in the main paper.

We first compare Baseline model with cosine decay learning rate to a Baseline model with Square-root learning rate decay that was used in the original paper (Tay et al., 2022). We observe that the model with square-root learning rate performs significantly worse compared to the model with cosine decay learning rate. Hence, we use model with cosine decay learning rate as our baseline.

Secondly, we try another schedule for Gradual Stacking, which contains 4 stages and is represented by 6-12-18-24. The stage lengths have been adjusted such that the average number of layers used over training is 20. We set the stage boundaries as 17k, 74k, and 172k respectively. We observe that 6-12-18-24 Gradual Stacking performs much better than 12-16-20-24 Gradual Stacking on various downstream tasks. This signals towards the dependency of Gradual Stacking on proper schedule selection.

On the other hand RAPTR doesn't show a big difference with 6-12-18-24 schedule, when compared to 12-16-20-24 schedule. A proper analysis of both the methods on schedule dependencies is kept for future work.

C SCHEDULE SELECTION

We follow two schedules from Reddi et al. (2023), Equal and Proportional, to build the lengths of each stage in RAPTR and Stacking. For Equal scheduling, used in Tables 1 and 2, we split training into k equal stages. For Proportional scheduling in Table 3, we increase the length of a stage in proportion to index of the stage. For example, for 12-16-20-24 RAPTR and Stacking with

proportional scheduling for $400k$ training steps, the stage lengths are set as $40k$, $80k$, and $120k$ respectively.

The flop counts are decided on the basis of average subnetwork length during training. For example, if the average subnetwork length is 18 for a model with 24 layers, we consider relative flops of RAPTR compared to baseline as 0.75.

For 12-16-20-24 RAPTR with Equal schedule, the average subnetwork length during training is given by 18, while for Proportional schedule, the average subnetwork length during training is given by 20. However, when either Equal and Proportional schedules don't return the target average subnetwork length, we re-consider the stage lengths by taking the closest candidate among Equal and Proportional schedules and remove equal steps from each of the first $k-1$ stages and add to the final full model training stage. For example, in table 4, with 6-12-18-24 RAPTR and target average subnetwork length 20, Proportional schedule returns an average subnetwork length of 18 and stage lengths $40k$, $80k$, $120k$ and $160k$ respectively. We then aim to find x such that stage lengths of $40k - x$, $80k - x$, $120k - x$ and $160k + 3x$ returns the desired average subnetwork length. On solving for x, we get $x = 22k$.

C.1 ABLATION ON DIFFERENT SCHEDULES FOR BERT-BASE

We conduct experiments on RAPTR with different stage schedules and same relative flop counts in table 1. We observe on BERT-base that different schedules differ by atmost 0.01 in pre-training loss and $0.1 - 0.5\%$ in downstream performance on average. Hence, we observe that RAPTR is robust to the stage schedule selection.

Hence, on UL2 we use the following stage schedules to minimize computational overhead. We restrict the number of phases in RAPTR to 4. The first stage selects a sub-network with half of the layers at random at each step. We increase the sub-network size by a particular amount in each stage transition, such that we train the full model in the final phase.

Table 5: Performance of RAPTR on BERT-base with different schedules. The average subnetwork size for each schedule is 9, which makes each schedule 1.33x faster than the baseline model (provided for reference). We use the same hyperparameters as used in table 1. We observe minor differences between different schedules, indicating robustness of RAPTR.

RAPTR schedule	Rel. flops	Eval loss	MNLI	QNLI	SST-2	Avg.
3-6-9-12	1	1.76	82.0	89.6	92.0	87.9
4-8-12	1	1.76	81.9	89.3	91.5	87.6
6-9-12	1	1.75	82.3	89.2	91.0	88.0
6-8-10-12	1	1.75	82.1	89.8	92.4	88.1
Baseline	1.33	1.76	81.5	90.5	91.4	87.8

C.2 ABLATION ON FIXED LAYER SET $\mathcal{I}$ FOR BERT-BASE

We conduct experiments on 6-8-10-12 RAPTR on BERT-base with different fixed layer set $\mathcal{I}$ (table 6). We observe that fixed set selection of $\{1, 12\}$ performs the best among other candidates. This selection however can be expensive to verify for other settings like UL2. Hence, in all subsequent experiments, we continue with the first and last layer fixed in all stages of RAPTR.

C.3 COMPARISON OF RAPTR TO PROGRESSIVE LAYERDROP (ZHANG & HE, 2020)

We implemented progressive layerdrop from (Zhang & He, 2020). The algorithm drops more layers towards the end of training. Kaddour et al. (2023) showed that such progressive drop rate can hurt the model's performance compared to baseline training and gradual stacking. We recover the observation by training a BERT-base model with progressive layerdrop and 6-8-10-12 RAPTR, both

Table 6: Performance of RAPTR on BERT-base with different $\mathcal{I}$ set for 6-8-10-12 RAPTR run for $100k$ steps. We use the same hyperparameters as used in table 1, except the training steps reduced to $100k$ steps. We observe that fixing the first and the last layer at all times lead to slightly better performance, compared to other candidates.

$\mathcal{I}$ set	Eval loss	MNLI	QNLI	SST-2	Avg.
$\{\}$	2.12	-	-	-	-
$\{1\}$	2.13	-	-	-	-
$\{1, 12\}$	**2.11**	78.6	86.77	88.69	84.7
$\{1, 2, 12\}$	2.13	-	-	-	-
$\{1, 11, 12\}$	2.12	77.4	87.0	89.4	84.6
$\{1, 2, 11, 12\}$	2.14	78.0	86.1	88.8	84.3
$\{1, 2, 3, 11, 12\}$	2.18	77.4	86.5	88.4	84.1
$\{1, 2, 10, 11, 12\}$	2.16	-	-	-	-

being 1.33 times faster compared to standard training. For a fair comparison, we tune the peak learning rate for both the methods and report the best performing model. We observe that at the end of training, progressive layerdrop is atleast 0.07 better in evaluation loss and 1% better on downstream performance on average.

Table 7: Comparison of RAPTR to progressive layerdrop (Zhang & He, 2020) on BERT-base. The peak learning rate for each method has been tuned for fair comparison and performance of the best model is reported. Here, we also report the baseline training in equiflop setting table 2 as reference. Progressive Layerdrop performs much worse than both the baseline and RAPTR runs, owing to it dropping more layers towards the end of training.

RAPTR schedule	Rel. flops	Eval loss	MNLI	QNLI	SST-2	Avg.
RAPTR	1.	1.73	83.1	90.7	91.0	88.3
Layerdrop (Zhang & He, 2020)	1.	1.8	81.7	89.0	90.8	87.2
Baseline	1.	1.74	82.4	90.5	91.8	88.2

Table 8: Comparison of RAPTR (with 30k initial full model training) from table 3 to progressive layerdrop (Zhang & He, 2020) on UL2. Both of the methods are run at peak learning rate of 10^{-2}. We set the average drop rate under progressive layerdrop to 0.83 so that the average FLOPs is 17% lower compared to a baseline run. Progressive Layerdrop performs much worse than RAPTR, owing to it dropping more layers towards the end of training.

RAPTR schedule	Trivia QA	Tydi QA	SQuADv2	SGLUE
RAPTR	25.8 (0.2)	36.7 (1.0)	44.1 (0.5)	60.9 (0.2)
Layerdrop (Zhang & He, 2020)	21.3 (0.3)	32.4 (2.1)	40.2 (0.9)	59.9 (0.2)

D SYSTEM SPEED OF DIFFERENT SUBNETWORKS

We report the speed of training in steps/sec for UL2 models with different subnetwork sizes on our hardware.

So, looking at table 3, let t be the total time for completing $400k$ steps in UL2 training. In the case of a 12-16-20-24 RAPTR with proportional stage lengths, it takes approximately $0.84t$ to complete $400k$ steps on our machine. For an ideal machine that perfectly realizes the theoretical speedups based on the desired flop count reduction, the total time required will be $0.83t$ (extremely close to what our current machine takes!).

Table 9: Training Steps/sec at different stages on 24-layer UL2

subnetwork size	steps/sec
12	2.0
16	1.6
20	1.4
24	1.1

We trained our BERT-base models on 4×4 TPU cores with percore batch size of 8. We report the speed of training in steps/sec with different subnetwork sizes.

Table 10: Training Steps/sec at different stages on BERT-base

subnetwork size	steps/sec
6	10.2
8	11.8
10	14.3
12	17.9

So, looking at table 2, let t be the total time for completing $675k$ steps in BERT-base training. In the case of a 6-8-10-12 RAPTR with Equal stage lengths, it takes approximately $0.79t$ to complete $675k$ steps on our machine. For an ideal machine that perfectly realizes the theoretical speedups based on the desired flop count reduction, the total time required will be $0.75t$.

E MORE EXTENSIVE DOWNSTREAM TESTS ON UL2 MODELS

We conduct extensive downstream evaluation of the trained models from table 3 on additional downstream tasks and under higher shot in-context setting. Additional tasks include QA tasks like natural QA (Kwiatkowski et al., 2019), web QA (Berant et al., 2013), DROP (Dua et al., 2019), CoQA (Reddy et al., 2019), and QuAC (Choi et al., 2018); and completion tasks like Storyclose (Mostafazadeh et al., 2017), Hellaswag (Zellers et al., 2019), and LAMBADA (Paperno et al., 2016).

F PROOF OF SCALING THEOREMS

For simplicity, we present the results for transformers with single head in the self-attention layer. Furthermore, for simplicity, we use σ_{relu} activation in the MLPs.

Algorithm 2: Transformer Layer

Require: 2 layer normalization layers f^{LN}_{attn}, f^{LN}_{mlp} (F.1), an MLP layer f^{mlp} (F.3), and a softmax self-attention layer f^{attn} (F.2), input sequence $\{\boldsymbol{x}_n \in \mathbb{R}^d\}_{n=1}^N$.

1: Attention Layer normalization: returns $\{\boldsymbol{y}_n^{attnln}\}_{n=1}^N$ with $\boldsymbol{y}_n^{attnln} = f^{LN}_{attn}(\boldsymbol{x}_n)$ for all $n \leq N$.
2: Softmax self-attention: returns $\{\boldsymbol{y}_n^{attn}\}_{n=1}^N = f^{attn}(\{\boldsymbol{y}_n^{attnln}\}_{n=1}^N)$.
3: Residual connection: returns $\{\boldsymbol{y}_n^{attnblock}\}_{n=1}^N$, with $\boldsymbol{y}_n^{attnblock} = \boldsymbol{x}_n + \boldsymbol{y}_n^{attn}$ for all $n \leq N$.
4: MLP Layer normalization: returns $\boldsymbol{y}_n^{mlpln} = f^{LN}_{mlp}(\boldsymbol{y}_n^{attnblock})$ for all $n \leq N$.
5: MLP function: returns $\{\boldsymbol{y}_n^{mlp}\}_{n=1}^N$ with $\boldsymbol{y}_n^{mlp} = f^{mlp}(\boldsymbol{y}_n^{mlpln})$ for all $n \leq N$.
6: Compute $\boldsymbol{y}_n = \boldsymbol{y}_n^{mlp} + \boldsymbol{y}_n^{attnblock}$ for all $n \leq N$.
7: **return** $\{\boldsymbol{y}_n\}_{n=1}^N$

Definition F.1. [Layer Normalization] Define a normalization function $f : \mathbb{R}^d \to \mathbb{R}^d$ that performs $f(\boldsymbol{x}) = (\boldsymbol{x} - \mu)/\sigma$, where μ and σ are the mean and standard deviation of $\boldsymbol{x}$, respectively. Then, layer normalization $f^{LN} : \mathbb{R}^d \to \mathbb{R}^d$ with parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^d$ takes as input $\boldsymbol{x} \in \mathbb{R}^d$ and outputs $\boldsymbol{y} \in \mathbb{R}^d$, which is computed as $\boldsymbol{z} = f(\boldsymbol{x}), \boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}$.

Table 11: We extensively compare all trained models from table 3 on multiple downstream tasks and few-shot in-context settings. We follow prompt design from Chowdhery et al. (2022) in each setting. For tasks marked with † and ⋆, we report Exact match and F1 scores respectively. For the rest, we use accuracy. The tasks have been sub-divided into 4 major groups, memorization QA, QA with context, completion, and SuperGLUE. On average, RAPTR is 1-2% better than baseline and stacking.

	1-shot				5-shot			
	Baseline	RAPTR	Stacking	RAPTR +init. train	Baseline	RAPTR	Stacking	RAPTR +init. train
Trivia QA†	25.0 (0.2)	22.2 (—)	20.1 (1.3)	25.8 (0.2)	26.5 (1.1)	23.4 (—)	21.1 (1.3)	25.1 (0.5)
Web QA†	6.4 (0.4)	5.9 (—)	5.8 (0.6)	7.6 (0.5)	10.6 (0.4)	8.2 (—)	9.2 (0.5)	11.2 (0.2)
Natural QA†	4.2 (0.5)	3.6 (—)	3.4 (0.4)	4.4 (0.1)	5.7 (0.1)	4.6 (—)	4.6 (0.3)	6.0 (0.3)
Tydi QA†	34.4 (3.1)	38.2 (—)	28.6 (2.7)	36.7 (1.0)	-	-	-	-
SQuaDv2†	42.1 (2.0)	40.6 (—)	36.0 (0.9)	44.1 (0.5)	43.2 (3.0)	40.0 (—)	36.2 (2.6)	44.5 (1.3)
DROP†	21.4 (0.8)	23.5 (—)	19.5 (0.6)	23.0 (0.4)	-	-	-	-
CoQA⋆	49.2 (0.7)	51.6 (—)	43.9 (0.8)	52.4 (0.7)	-	-	-	-
QuAC⋆	18.1 (0.5)	18.8 (—)	16.8 (0.6)	18.1 (0.4)	-	-	-	-
LAMBADA	13.7 (2.9)	12.6 (—)	12.0 (1.1)	18.7 (3.1)	30.5 (3.4)	32.0 (—)	29.5 (2.1)	38.7 (2.2)
Storycloze	72.9 (0.4)	73.5 (—)	71.0 (0.4)	73.3 (0.4)	75.1 (0.1)	75.5 (—)	72.6 (0.8)	75.3 (0.6)
Hellaswag	58.3 (0.2)	57.5 (—)	56.1 (0.1)	58.5 (0.3)	58.3 (0.2)	57.3 (—)	56.1 (0.1)	58.4 (0.3)
SuperGLUE	60.0 (0.4)	60.1 (—)	60.4 (0.9)	60.9 (0.2)	60.7 (0.3)	60.6 (—)	58.8 (0.5)	62.1 (0.2)
Average	33.8	34.1	31.1	35.3	38.8	37.7	36.0	40.2

Definition F.2 (Softmax self-attention)**.** A self-attention layer $f^{attn} : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ with parameters $\{\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V, \boldsymbol{C}^{attn} \in \mathbb{R}^{d\times d}\}$ takes a sequence $\{\boldsymbol{x}_n\}_{n\le N}$ and outputs a sequence $\{\boldsymbol{y}_n\}_{n\le N}$, such that

$$\boldsymbol{y}_n = \boldsymbol{C}^{attn} \sum_{\overline{n}=1}^{N} a_{n,\overline{n}} \boldsymbol{v}_{\overline{n}},$$
$$\text{with } a_{n,\overline{n}} = \text{softmax}(\boldsymbol{K}\boldsymbol{q}_n)_{\overline{n}}, \quad \boldsymbol{q}_n = \boldsymbol{W}_Q \boldsymbol{x}_n, \quad \boldsymbol{k}_n = \boldsymbol{W}_K \boldsymbol{x}_n, \quad \boldsymbol{v}_n = \boldsymbol{W}_V \boldsymbol{x}_n,$$

for all $n \le N$, and $\boldsymbol{K} \in \mathbb{R}^{N\times N}$ defined with rows $\{\boldsymbol{k}_n\}_{n=1}^N$.

Definition F.3 (MLP)**.** An MLP layer $f^{mlp} : \mathbb{R}^d \to \mathbb{R}^d$ with parameters $\{\boldsymbol{W} \in \mathbb{R}^{m\times d}, \boldsymbol{C}^{mlp} \in \mathbb{R}^{m\times d}\}$ and activation σ_{relu}, takes an input $\boldsymbol{x} \in \mathbb{R}^d$ and outputs $\boldsymbol{y} \in \mathbb{R}^d$, such that

$$\boldsymbol{y} = \boldsymbol{C}^{mlp} \sigma_{relu}(\boldsymbol{W}\boldsymbol{x}).$$

Definition F.4 (Transformer layer)**.** A pre-layernorm transformer layer with three sub-layers; 2 layer normalization layers f^{LN}_{attn}, f^{LN}_{mlp} (F.1), an MLP layer f^{mlp} (F.3), and a softmax self-attention layer f^{attn} (F.2); takes a sequence $\{\boldsymbol{x}_n\}_{n\le N}$ and outputs a sequence $\{\boldsymbol{y}_n\}_{n\le N}$ in four steps.

1. First computes $\{\boldsymbol{y}^{attnln}_n\}_{n=1}^N$ using a layer normalization, i.e. $\boldsymbol{y}^{attnln}_n = f^{LN}_{attn}(\boldsymbol{x}_n)$ for all $n \le N$.
2. Then it runs softmax self-attention to get $\{\boldsymbol{y}^{attn}_n\}_{n=1}^N = f^{attn}(\{\boldsymbol{y}^{attnln}_n\}_{n=1}^N)$.
3. The net output of the self-attention block is given by $\{\boldsymbol{y}^{attnblock}_n\}_{n=1}^N$, with $\boldsymbol{y}^{attnblock}_n = \boldsymbol{x}_n + \boldsymbol{y}^{attn}_n$ for all $n \le N$.
4. Before passing to the MLP function, it is passed through another layer normalization, i.e. $\boldsymbol{y}^{mlpln}_n = f^{LN}_{mlp}(\boldsymbol{y}^{attnblock}_n)$ for all $n \le N$.
5. MLP function then returns $\{\boldsymbol{y}^{mlp}_n\}_{n=1}^N$ with $\boldsymbol{y}^{mlp}_n = f^{mlp}(\boldsymbol{y}^{mlpln}_n)$ for all $n \le N$.
6. $\boldsymbol{y}_n = \boldsymbol{y}^{mlp}_n + \boldsymbol{y}^{attnblock}_n$ for all $n \le N$.

Definition F.5 (Initialization of the weights in the transformer layer)**.** The weights are initialized as follows:

$$\boldsymbol{C}^{mlp}, \boldsymbol{C}^{attn}, \boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \sim \mathcal{N}(\mathbf{0}, \frac{1}{\sqrt{d}}\mathbf{I}),$$

$$\boldsymbol{W} \sim \mathcal{N}(\mathbf{0}, \frac{\sqrt{2}}{\sqrt{m}}\mathbf{I}).$$

The parameters $\gamma, \boldsymbol{b}$ of the functions $f_{attn}^{LN}, f_{mlp}^{LN}$ have been initialized as $\mathbf{1}$ and $\mathbf{0}$ respectively.

Lemma F.6 (Norm of the output of the MLP function, modification of lemma 7.1 in Allen-Zhu et al. (2019))**.** *For a given input sequence $\{\boldsymbol{y}_n^{mlpln}\}_{n\leq N}$, if $\varepsilon \in (0,1]$, with probability at least $1 - O(N)\cdot e^{-\Omega\left(m\varepsilon^2\right)}$ over the randomness of $\boldsymbol{C}^{mlp}, \boldsymbol{W}$, we have*

$$\forall i \in [N]: \quad \left\|\boldsymbol{y}_n^{mlp}\right\| / \left\|\boldsymbol{y}_n^{mlpln}\right\| \in [1-\varepsilon, 1+\varepsilon].$$

Lemma F.7 (Norm of the output of the layer normalization layers)**.** *Given any input sequence $\{\boldsymbol{x}_n\}_{n=1}^N$, under the assumption for all $n \leq N$, $\boldsymbol{x}_n - \sum_{i=1}^d x_{n,i}$ isn't identically $\mathbf{0}$, we have*

$$\left\|\boldsymbol{y}_n^{attnln}\right\| = \sqrt{d}$$

for all $n \leq N$.

Similar result holds for $\{\boldsymbol{y}_n^{mlpln}\}_{n=1}^N$.

Lemma F.8 (Norm of the output of the MLP block)**.** *For a given input sequence $\{\boldsymbol{y}_n^{attnblock}\}_{n\leq N}$, with probability at least $1 - \mathcal{O}(1)$ over the randomness of $\boldsymbol{C}^{mlp}, \boldsymbol{W}$, we have*

$$\begin{aligned}\left\|\boldsymbol{y}_n\right\|^2 = \left\|\boldsymbol{y}_n^{mlp} + \boldsymbol{y}_n^{attnblock}\right\|^2 = \left\|\boldsymbol{y}_n^{attnblock}\right\|^2 + d \\ + \mathcal{O}\left(\left\|\boldsymbol{y}_n^{attnblock}\right\| \log N + (d + \left\|\boldsymbol{y}_n^{attnblock}\right\|)\frac{\log^{3/2} N}{\sqrt{m}}\right),\end{aligned}$$

for all $n \leq N$.

Proof. Combining lemmas F.6 and F.7, we have for the sequence $\{\boldsymbol{y}_n^{attnblock}\}_{n\leq N}$,

$$\left|\left\|\boldsymbol{y}_n^{mlp}\right\| / \sqrt{d} - 1\right| \leq \mathcal{O}\left(\frac{\sqrt{\log N}}{\sqrt{m}}\right),$$

w.p. atleast $1 - \mathcal{O}(1)$.

Furthermore, due to the randomness of $\boldsymbol{C}^{mlp}$, we can further show that w.p. at least $1 - \mathcal{O}(1)$,

$$|\langle \boldsymbol{y}_n^{attnblock}, \boldsymbol{y}_n^{mlp}\rangle| \leq \mathcal{O}(\left\|\boldsymbol{y}_n^{attnblock}\right\| \log N),$$

for all $n \leq N$. Combining the two results, we have

$$\begin{aligned}\left\|\boldsymbol{y}_n\right\|^2 = \left\|\boldsymbol{y}_n^{mlp} + \boldsymbol{y}_n^{attnblock}\right\|^2 = \left\|\boldsymbol{y}_n^{attnblock}\right\|^2 + d \\ + \mathcal{O}\left(\left\|\boldsymbol{y}_n^{attnblock}\right\| \log N + (d + \left\|\boldsymbol{y}_n^{attnblock}\right\|)\frac{\log^{3/2} N}{\sqrt{m}}\right).\end{aligned}$$

□

Lemma F.9 (Norm of the output of the attention block)**.** *For a given input sequence $\{\boldsymbol{x}_n\}_{n\leq N}$, if $\varepsilon \in (0,1]$, with probability at least $1 - O(1)$ over the randomness of $\boldsymbol{C}^{attn}, \boldsymbol{W}_V$, we have*

$$\forall i \in [N]: \quad \left\|\boldsymbol{x}_n\right\|^2 \leq \left\|\boldsymbol{y}_n^{attnblock}\right\|^2 \leq \left\|\boldsymbol{x}_n\right\|^2 + \left\|\boldsymbol{x}_n\right\| + d + \mathcal{O}(\frac{\sqrt{\log N}}{\sqrt{d}}(d + \left\|\boldsymbol{x}_n\right\|)).$$

Proof. With the randomness of $\boldsymbol{C}^{attn}$ and $\boldsymbol{W}_V$, we have

$$\left|\frac{\left\|\boldsymbol{y}_n^{attn}\right\|}{\left\|\sum_{j\leq N} a_{n,j}\boldsymbol{y}_j^{attnln}\right\|} - 1\right| \leq \mathcal{O}\left(\frac{\sqrt{\log N}}{\sqrt{d}}\right),$$

for all $n \leq N$ w.p. atleast $1 - \mathcal{O}(1)$.

Furthermore, due to the randomness of $\boldsymbol{C}^{attn}$, we can further show that w.p. atleast $1 - \mathcal{O}(1)$,

$$|\langle \boldsymbol{y}_n^{attn}, \boldsymbol{x}_n \rangle| \leq \mathcal{O}(\|\boldsymbol{x}_n\| \log N),$$

for all $n \leq N$.

Using Cauchy-Schwarz inequality, we must have

$$\left\| \sum_{j \leq N} a_{n,j} \boldsymbol{y}_j^{attnln} \right\| \leq \max_{j \leq N} \left\| \boldsymbol{y}_j^{attnln} \right\| = \sqrt{d}.$$

Thus, combining the results, we have

$$\begin{aligned} \left\| \boldsymbol{y}_n^{attnblock} \right\|^2 = \left\| \boldsymbol{y}_n^{attn} + \boldsymbol{x}_n \right\|^2 \leq \|\boldsymbol{x}_n\|^2 + d \\ + \mathcal{O}\left(\|\boldsymbol{x}_n\| \log N + (d + \|\boldsymbol{x}_n\|) \frac{\log^{3/2} N}{\sqrt{d}} \right). \end{aligned}$$

□

Lemma F.10 (Norm of the output of the transformer layer)**.** *For a given input sequence $\{\boldsymbol{x}_n\}_{n \leq N}$, if $\varepsilon \in (0, 1]$, with probability at least $1 - \mathcal{O}(1)$ over the randomness of $\boldsymbol{C}^{attn}, \boldsymbol{W}_V, \boldsymbol{C}^{mlp}$, we have*

$$\forall i \in [N]: \quad \|\boldsymbol{x}_n\|^2 + d + \mathcal{O}(err) \leq \|\boldsymbol{y}_n\|^2 \leq \|\boldsymbol{x}_n\|^2 + 2d + \mathcal{O}(err),$$

where $err = \mathcal{O}\left(\|\boldsymbol{x}_n\| \log N + (d + \|\boldsymbol{x}_n\|) \frac{\log^{3/2} N}{\sqrt{m}} \right)$.

Theorem F.11 (Square-root scaling of input norms)**.** *At initialization, given any input sequence $\boldsymbol{x}_{1:N}$ and scalars $\alpha_{1:L}$, w.h.p. the intermediate sequences of F (definition 2.1) satisfy*

$$\left\| \boldsymbol{y}_i^{(\ell)} \right\|_2^2 = \|\boldsymbol{x}_i\|_2^2 + \Theta(\sum_{j=1}^{\ell} \alpha_j^2 d), \quad \textit{for all } 1 \leq i \leq N, 1 \leq \ell \leq L.$$

Proof. This follows from the fact that the transformer architecture is a stack of struturally identical L transformer layers, and each transformer layer's output norm increases by $\Theta(d)$ compared to its input norm, as given by lemma F.10. □

G NOISE STABILITY FOR TRANSFORMERS

Theorem G.1. *Suppose $\mathcal{L}$ satisfies the following condition $\mathcal{L}$ satisfies the following condition for any input $\tilde{\boldsymbol{x}}, \eta \in \mathbb{R}^d$ and label $\boldsymbol{v} \in \{0, 1\}^V$ for some constant μ_{loss} :*

$$\mathcal{L}(\tilde{\boldsymbol{x}} + \eta \|\tilde{\boldsymbol{x}}\|, \boldsymbol{v}) - \mathcal{L}(\tilde{\boldsymbol{x}}, \boldsymbol{v}) \leq \mu_{loss} \|\eta\|_2 . \tag{1}$$

The difference between expected loss of $(L-1)$-RAPTR and L-RAPTR, i.e. $|\mathcal{L}_2(F) - \mathcal{L}_1(F)|$, is upper bounded by $\frac{C}{L} \mathbb{E}_{\boldsymbol{x} \in D}(\sum_{\ell=1}^{L} \Psi_\ell(\boldsymbol{x})) / \|F(\boldsymbol{x})\|_2$ for some constant C that depends on regularity properties of the head H on top of the backbone F.

Proof of theorem 4.2. Denote by $F_{-\ell}$ the L candidates that we can randomly choose from during $L-1$ RAPTR training. The output of $F_{-\ell}$ on any input $\boldsymbol{x}$ is equivalent to the output of the model F on $\boldsymbol{x}$ and $\alpha_{1:L}$, with $\alpha_\ell = 0$ and the rest set to 1.

Then, the average loss of $L-1$ random training is given by

$$\mathcal{L}_2(F) = \frac{1}{L} \sum_{i=1}^{L} \mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \sim D} \mathcal{L}\left(F_{-\ell}(\boldsymbol{x}), \boldsymbol{v}\right).$$

On the other hand, the loss of the full model is given by

$$\mathcal{L}_1(F) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{v}\sim D}\mathcal{L}\left(F(\boldsymbol{x}),\boldsymbol{v}\right).$$

Hence, the expected difference between loss of $L-1$ random training and full model is given by

$$\begin{aligned}
\mathcal{L}_2(F) - \mathcal{L}_1(F) &= \frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{\boldsymbol{x},\boldsymbol{v}\sim D}\left(\mathcal{L}\left(F_{-\ell}(\boldsymbol{x}),\boldsymbol{v}\right) - \mathcal{L}\left(F(\boldsymbol{x}),\boldsymbol{v}\right)\right) \\
&\leq \frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{\boldsymbol{x},\boldsymbol{v}\sim D} \quad \mu_{loss}\frac{\|F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x})\|_2}{\|F(\boldsymbol{x})\|_2} \\
&\leq \frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{\boldsymbol{x},\boldsymbol{v}\sim D} \quad \mu_{loss}\frac{\psi_\ell}{\|F(\boldsymbol{x})\|_2}.
\end{aligned}$$

Here the pre-final step uses the definition of μ_{loss} from eq. (1), and the final step uses the definition of Ψ_ℓ from definition 4.1. □

Discussion An example of a loss function with bounded μ_{loss} is a transformer model, that uses a layer normalization layer before using an embedding matrix Φ to compute the logits. The logits are then compared with the true label with a cross-entropy loss with a softmax function on logits. $\psi(\mathcal{L})$ hence depends on the ℓ_2 norm of Φ and the weight parameters of the layer normalization layer.

G.1 Noise stability of linear residual networks

Proof of lemma 4.3. We outline the proof for each case.

1. With residual connection and layer normalization, the function F computes the intermediate activations $\boldsymbol{y}^{(1)},\cdots,\boldsymbol{y}^{(L)}$ on an input $\boldsymbol{x}$ as follows.

$$\boldsymbol{y}^{(\ell)} = \left(\boldsymbol{I} + \frac{\boldsymbol{W}_\ell}{\|\boldsymbol{y}^{(\ell-1)}\|}\right)\boldsymbol{y}^{(\ell-1)}.$$

Similar to lemma F.6, we can show w.h.p. with respect to the randomness of $\boldsymbol{W}_1,\cdots,\boldsymbol{W}_L$, for all $\ell \leq L$, $\sqrt{\ell}(1-\mathcal{O}(1)) \leq \left\|\boldsymbol{y}^{(\ell)}\right\|_2 \leq \sqrt{\ell}(1+\mathcal{O}(1))$.

Ignoring the error terms, we can simply write

$$\boldsymbol{y}^{(\ell)} = \left(\boldsymbol{I} + \frac{\boldsymbol{W}_\ell}{\sqrt{\ell}}\right)\boldsymbol{y}^{(\ell-1)}.$$

Pick a general layer ℓ to be dropped. Then,

$$F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x}) = \prod_{\ell'=\ell+1}^{L}\left(1+\frac{\boldsymbol{W}_{\ell'}}{\sqrt{\ell'}}\right)\frac{\boldsymbol{W}_\ell}{\sqrt{\ell}}\boldsymbol{y}^{(\ell-1)} + err,$$

where the err term appears because of the change in scales of activations $\boldsymbol{y}^{(\ell+1)},\cdots,\boldsymbol{y}^{(L)}$ with the dropping of layer ℓ. This error can be bounded as $\mathcal{O}(\sqrt{L}/\ell)$ using the same procedure followed below to bound the first term on R.H.S..

Similar to bounding the norms of $\boldsymbol{y}^{(\ell)}$, w.h.p. we can show that

$$\left\|\prod_{\ell'=\ell+1}^{L}\left(1+\frac{\boldsymbol{W}_{\ell'}}{\sqrt{\ell'}}\right)\frac{\boldsymbol{W}_\ell}{\sqrt{\ell}}\boldsymbol{y}^{(\ell-1)}\right\|_2 \leq \sqrt{\frac{L}{\ell}}.$$

Hence,

$$\psi_\ell := \|F_{-\ell}(\boldsymbol{x}) - F_\ell(\boldsymbol{x})\|_2 \leq \mathcal{O}(\sqrt{L/\ell}).$$

This implies

$$\frac{1}{L}\sum_{\ell}\psi_\ell(\boldsymbol{x}) = \frac{1}{L}\sum_{\ell}\mathcal{O}(\sqrt{L/\ell}) = \mathcal{O}(1).$$

Since the gap in $\mathcal{L}_2$ and $\mathcal{L}_1$ is bounded by $\mathcal{O}(\mathbb{E}_{\boldsymbol{x}}\frac{1}{L}\sum_\ell \psi_\ell(\boldsymbol{x})/||F(\boldsymbol{x})||)$ from theorem 4.2, we have the gap as $\mathbf{O}\left(\frac{1}{\sqrt{L}}\right)$.

2. With no normalization, the function F looks as follows.

$$\boldsymbol{y}^{(\ell)} = (\boldsymbol{I} + \boldsymbol{W}_\ell)\,\boldsymbol{y}^{(\ell-1)}.$$

Similar to lemma F.6, we can show w.h.p. with respect to the randomness of $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$, for all $\ell \le L$, $2^{\ell/2}(1-\mathcal{O}(1)) \le \left\|\boldsymbol{y}^{(\ell)}\right\|_2 \le 2^{\ell/2}(1+\mathcal{O}(1))$.

With a drop in layer, we get

$$F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x}) = \prod_{\ell'=\ell+1}^{L}\left(\boldsymbol{I}+\boldsymbol{W}^{\ell'}\right)\boldsymbol{W}_\ell\prod_{\ell'=1}^{\ell-1}\left(\boldsymbol{I}+\boldsymbol{W}^{\ell'}\right)\boldsymbol{x}.$$

Similar to lemma F.6, we can show w.h.p. with respect to the randomness of $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$, $\left\|\prod_{\ell'=\ell+1}^{L}\left(\boldsymbol{I}+\boldsymbol{W}^{\ell'}\right)\boldsymbol{W}_\ell\prod_{\ell'=1}^{\ell-1}\left(\boldsymbol{I}+\boldsymbol{W}^{\ell'}\right)\boldsymbol{x}\right\|_2 \ge \mathcal{O}(2^{(\ell-1)/2})$.

This implies $\|F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x})\|_2 = \Omega(2^{(\ell-1)/2})$.

The argument can now be completed by using theorem 4.2.

3. Without residual connection, the function F looks as follows.

$$\boldsymbol{y}^{(\ell)} = \boldsymbol{W}_\ell\frac{\boldsymbol{y}^{(\ell-1)}}{\left\|\boldsymbol{y}^{(\ell-1)}\right\|_2}.$$

Similar to lemma F.6, we can show w.h.p. with respect to the randomness of $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$, for all $\ell \le L$, $(1-\mathcal{O}(1)) \le \left\|\boldsymbol{y}^{(\ell)}\right\|_2 \le (1+\mathcal{O}(1))$. Thus, ignoring error terms, the network F roughly looks like

$$\boldsymbol{y}^{(\ell)} = \boldsymbol{W}_\ell\boldsymbol{y}^{(\ell-1)}.$$

With a drop in layer, we get

$$F_{-\ell}(\boldsymbol{x}) - F(\boldsymbol{x}) = \prod_{\ell'=\ell+1}^{L}\boldsymbol{W}_{\ell'}(\boldsymbol{W}^{(\ell)}-\boldsymbol{I})\prod_{\ell'=1}^{\ell-1}\boldsymbol{W}_{\ell'}\boldsymbol{x}.$$

Using randomness of $\boldsymbol{W}^{(\ell)}$ this can be shown to be of norm $\Omega(1)$. The argument can now be completed by using theorem 4.2.

□

Lemma G.2. *When the layers are perfectly aligned, i.e. all weights $\boldsymbol{W}_i = \boldsymbol{A}$ for some matrix $\boldsymbol{A}$ with $\|\boldsymbol{A}\|_2 = 1$ and for simplicity, assume second eigenvalue $\lambda_2(\boldsymbol{A}) = 1-\delta$ for some $\delta > 0$, we have*

(a) With residual connection & layernorm, we have $\Psi_\ell(\boldsymbol{x}) = \mathcal{O}(1)$ and $\|F(\boldsymbol{x})\| = \Omega(L)$. Thus, the gap in losses between stages is $\mathcal{O}(1/L)$.

(b) Without residual connection, we have both $\Psi_\ell(\boldsymbol{x}) = \mathcal{O}(\delta^{-1}e^{-\ell})$ and $\|F(\boldsymbol{x})\| = \mathcal{O}(1)$. Thus the gap in losses between stages is $\mathcal{O}(\frac{1}{L})$.

(c) Without layernorm, we have $\Psi_\ell(\boldsymbol{x}) = \Omega(2^{L-1})$ *and* $\|F(\boldsymbol{x})\| = \mathcal{O}(2^L)$ *. Thus the gap in losses between stages is* $\Omega(1)$.

Proof of lemma G.2. We outline the proof for each case.

1. With residual connection & layernorm, the function F computes the intermediate activations $\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(L)}$ on an input $\boldsymbol{x}$ as follows.

$$\boldsymbol{y}^{(\ell)} = \left(\boldsymbol{I} + \frac{\boldsymbol{A}}{\left\|\boldsymbol{y}^{(\ell-1)}\right\|}\right)\boldsymbol{y}^{(\ell-1)}.$$

   We show that the above method will behave like power method, i.e. $\boldsymbol{y}^{(\ell)}$ will be of magnitude $\Omega(\ell)$ and will be ϵ-close in angle to the top eigenvector of $\boldsymbol{A}$, denoted as $\boldsymbol{v}_1(\boldsymbol{A})$, provided $\ell \geq \Omega((1/\delta)\log(1/\epsilon))$.

   We give an inductive argument as follows. Consider any layer ℓ. Say θ_ℓ denotes the angle between $\boldsymbol{y}^{(\ell)}$ and $\boldsymbol{v}_1(\boldsymbol{A})$. Also, say $\Pi^{\perp}_{\boldsymbol{v}1}$ denotes an orthogonal projection to subspace spanned by the rest of the eigenvectors of $\boldsymbol{A}$. Then,

$$\begin{aligned}
|\tan\theta_{\ell+1}| &= \frac{|\langle \boldsymbol{v}_1(\boldsymbol{A}), \boldsymbol{y}^{(\ell+1)}\rangle|}{\left\|\Pi^{\perp}_{\boldsymbol{v}1}(\boldsymbol{y}^{(\ell+1)})\right\|_2} \\
&= \frac{|\langle \boldsymbol{v}_1(\boldsymbol{A}), \boldsymbol{y}^{(\ell)}\rangle + \langle \boldsymbol{v}_1(\boldsymbol{A}), \boldsymbol{A}\boldsymbol{y}^{(\ell)}\rangle|}{\left\|\Pi^{\perp}_{\boldsymbol{v}1}(\boldsymbol{y}^{(\ell)}) + \Pi^{\perp}_{\boldsymbol{v}1}\boldsymbol{A}\boldsymbol{y}^{(\ell)}\right\|_2} \\
&= \frac{2|\langle \boldsymbol{v}_1(\boldsymbol{A}), \boldsymbol{y}^{(\ell)}\rangle|}{\left\|\Pi^{\perp}_{\boldsymbol{v}1}(\boldsymbol{y}^{(\ell)}) + \Pi^{\perp}_{\boldsymbol{v}1}\boldsymbol{A}\boldsymbol{y}^{(\ell)}\right\|_2} \\
&\leq \frac{2\langle \boldsymbol{v}_1(\boldsymbol{A}), \boldsymbol{y}^{(\ell)}\rangle}{\left\|\Pi^{\perp}_{\boldsymbol{v}1}(\boldsymbol{y}^{(\ell)}) + \lambda_2(\boldsymbol{A})\Pi^{\perp}_{\boldsymbol{v}1}\boldsymbol{y}^{(\ell)}\right\|_2} \\
&= \frac{2}{1+\lambda_2(\boldsymbol{A})}|\tan\theta_\ell|.
\end{aligned}$$

   This implies, under $\lambda_2(\boldsymbol{A}) < 1$, $|\tan\theta_\ell|$ decreases with ℓ. Under the assumption that $\boldsymbol{x}$ isn't orthogonal to $\boldsymbol{v}_1(\boldsymbol{A})$, the above condition simply shows that if $\ell \geq \mathcal{O}((1/\delta)\log(1/\epsilon))$, than $|\tan\theta_\ell| < \epsilon$.

   Furthermore, once aligned (or closely aligned) to $\boldsymbol{v}_1(\boldsymbol{A})$, the norm of $\boldsymbol{y}^{(\ell)}$ grows linearly. Hence, $\left\|\boldsymbol{y}^{(\ell)}\right\| = \Omega(L)$. Furthermore, for any ℓ, the gap between $F_{-\ell}$ and F_ℓ simply becomes equal to the gap between L and $L-1$ steps of the modified power method, which can be bounded as $\mathcal{O}(1)$.

2. With no residual connection, the function F computes the intermediate activations $\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(L)}$ on an input $\boldsymbol{x}$ as follows.

$$\boldsymbol{y}^{(\ell)} = \frac{\boldsymbol{A}}{\left\|\boldsymbol{y}^{(\ell-1)}\right\|}\boldsymbol{y}^{(\ell-1)}.$$

   From the update, it's trivial to come to the conclusion that $\boldsymbol{y}^{(\ell)}$ will stay unit norm for any layer ℓ.

   This update is exactly power-method, where $\boldsymbol{y}^{(\ell)}$ gets ϵ-close to the top eigenvector direction of $\boldsymbol{A}$ in $\mathcal{O}((1/\delta)\log(1/\epsilon))$ steps. The result then follows from bounding the gap between L and $L-1$ steps of power-method.

3. The proof is very similar to the proof of lemma 4.3 (3), where the major difference comes from the blowup in the norms of the intermediate activations.

□