
Improving Abstractive Summarization with Unsupervised Dynamic LoRA Mixtures

D. I. Chernyshev^{1,*}

(Submitted by A. A. Editor-name)

¹*Research Computing Center, Lomonosov Moscow State University, Leninskie Gory, 1, Moscow, 119234 Russia*

Received June 13, 2023

Abstract—The recent jump in hardware requirements for state-of-the-art language models have increased training and inference costs considerably. While methods such as Mixture-of-Experts (MoE) address the network throughput decline they retain high memory consumption. To alleviate the issue researches explored the possibility of replicating MoE results with layer adapters such as LoRA using various ensembling schemes. However, the utilization of the pre-trained LoRA adapters in the current approaches is limited as they are only considered in static ensembles while the dynamic counterparts are designed to be trained from scratch. We argue that dynamic LoRA ensembles can be obtained without additional fine-tuning by leveraging the network inference statistics. We propose Unsupervised Dynamic LoRA Mixtures (UDLM), a dynamic adapter ensembling method that translates pre-trained LoRA adapter sets to equivalent MoE networks by accumulating adapter spatial statistics. Evaluation on 6 abstractive summarization datasets in knowledge transfer setting demonstrates that UDLM substantially outperforms best static ensembles having a 30% summarization quality improvement while introducing only 35% increased latency.

2010 Mathematical Subject Classification: 68T50

Keywords and phrases: *Neural Networks, Abstractive Summarization, Machine Learning*

1. INTRODUCTION

Recent advances in language modeling has enabled creation of universal task-solvers such as ChatGPT. However, the power of those models come not from architectural specificities or complex training procedures but from the experimental scaling of model parameters and training data. Large language models (LLM) contain billions of parameters that were tuned on terabytes of textual data which gives them exceptional language comprehension abilities such as in-context learning or multi-step reasoning. At the same time the scale of state-of-the-art LLMs imposes strict hardware requirements and high inference costs which limits their applicability.

To cut the inference costs researchers [2, 3, 10] explored an alternative model scaling scheme, Mixture-of-Experts [1] (MoE), that unlike traditional layer-stacking scaling, increases the model capacity by ensembling layer feed-forward blocks (called experts) which enables better computation parallelization and efficient parameter offloading. For instance, by exploiting the MoE scheme Mixtral [2] LLM retains the network throughput of 13B model while having the capacity of 47B parameters. To reduce the hardware memory requirements the researchers [7, 8] explored adapter-tuning methods such as LoRA [6] and employed static adapter ensembling techniques [8, 11, 12] that replicate the results of full MoE networks. While several works [9, 17, 18, 30] considered applying LoRA for training a memory-efficient MoE models to our best knowledge there are none investigating the possibility of unsupervised dynamic ensembling of the pre-trained LoRA task adapters.

To address the lack of adapter-based MoE plug-and-play approaches we propose Unsupervised Dynamic LoRA Mixtures (UDLM), a dynamic ensembling method that translates pre-trained LoRA

* E-mail: chdanorbis@yandex.ru

adapter sets to equivalent MoE network. Unlike previous ensembling approaches our method doesn't rely on any tuning procedures and instead is initialized by accumulating the adapter inference statistics which enables inference-time ad-hoc corrections. Evaluation on 6 summarization datasets demonstrates that UDLM significantly outperforms best static LoRA ensembling methods achieving up-to 30% and 5.4% average ROUGE improvements for knowledge transfer and knowledge boosting settings respectively. Performance benchmark proves UDLM compatible with low-resource environments having only 20% increase in peak memory usage and 35% higher latency over the regular LoRA.

2. RELATED WORK

Low Rank adapter ensembling have been studied for both static (Model Soup) and dynamic (Mixture-of-Experts) settings. Chronopoulou et al. [11] proposed applying the original Model Soup [12] method directly to LoRA [6] models for GPT-2 [19] ad-hoc domain adaptation by averaging (uniformly) and merging the pre-trained adapters with highest similarity to the target domain. Evaluation on 10 datasets demonstrated that adapter soup approach has much higher efficiency than single best adapter especially on domains that have the lowest similarity to the training set. The approach was further refined by Huang et al. [8] in LoraHub method which employs gradient-free adapter weight optimization scheme to maximize the efficiency of model-adapter merging for downstream tasks. By fine-tuning the adapter merging weights LoraHub ensembles managed to match the quality of dedicated task-tuned LoRA adapters, thus demonstrating the viability of LoRA ensembling in low-resource environment. A gradient-based adapter weight optimization was explored by Lv et al. [13], however they found that non-gradient methods such as linear programming have better efficiency and robustness.

Another line of work utilized parameter arithmetic [14] approach to compose LoRA models with desired traits. Zhang et al. [15] showed that applying parameter arithmetic operations directly to LoRA adapters can improve domain understanding, increase model universality and reduce the data-induced bias such as toxicity. By negating the adapters trained on toxic content authors managed to reduce the toxicity scores of GPT-2 [19] model tenfold while keeping the language perplexity loss marginal. Similarly, Chronopoulou et al [16] showed that applying parameter arithmetic over language-tuned LoRA adapters can improve single-language downstream task performance of the merged model as well as enables cross-lingual knowledge transfer, which during evaluation on multi-lingual news summarization outperformed traditional fine-tuning for low-resource languages.

Wang et al. [9] proposed a hybrid approach, AdaMix, that leverages Mixture-of-Expert scheme for LoRA joint multi-adapter training and then merges the trained adapters in Model Soup fashion. Benchmark of BERT [20] and GPT-2 [19] training methods showed that AdaMix outperforms other parameter-efficient tuning schemes including LoRA. Ponti et al. [17] refined the approach for layer-wise merging by additionally learning adapter merging weights for each task separately. Full token-level Mixture-of-Expert for LoRA large language model training was tested by Dou et al. [18] and was shown to achieve significant improvement over base model fine-tuning on world knowledge benchmarks.

3. METHODOLOGY

Unsupervised Dynamic LoRA Mixtures (UDLM) can be conceptualized as a modification of Mixture-of-Expert (MoE) network where experts are decomposed through LoRA and gating function is defined as adapter semantic similarity search. In this section we first provide an overview of the base components and then describe the details of our unsupervised MoE initialization strategy.

3.1. Mixture-of-Experts

Mixture-of-Experts (MoE) [1] is a neural network scaling approach that preserves the layer-wise depth of the original architecture by replacing the feed-forward blocks with feed-forward ensembles (denoted as experts) in existing layers. While there are many implementations of MoE mechanism [5] recent state-of-the-art large language models [2, 3, 10] follow sparse MoE architecture [4] that aims to retain the original network throughput by limiting the maximum number of active Experts in each layer.

Formally, let N be the duplication factor and $\{E_1, E_2, \dots, E_N\} \in \mathbb{R}^{d_{in} \times d_{out}}$ the feed-forward expert blocks. Let $G: \mathbb{R}^{d_{in}} \mapsto \mathbb{R}^N$ be the gating function that assigns weights $g_i \geq 0$ to each expert E_i output and $\|g\| = 1$. For the layer input $h \in \mathbb{R}^{m \times d_{in}}$ (network hidden state) of the original feed-forward block the output of corresponding MoE expansion is defined as:

$$MoE(h) = \sum_{i=1}^N g_i \cdot hE_i \quad (1)$$

To limit the number of active experts E_i and thus save the computational costs the gating vector g is sparsified by masking the non-essential components. The most popular variant [2] of G is implemented by taking the softmax over the Top-K logits of a router matrix $W_G \in \mathbb{R}^{d_{in} \times N}$:

$$g = G(h) = \text{Softmax}(\text{TopK}(h \cdot W_G))$$

where $(\text{TopK}(x))_i := x_i$ if x_i is among the top-K coordinates of logits $x \in \mathbb{R}^N$ and $(\text{TopK}(x))_i := -\infty$ otherwise.

Essentially the K hyperparameter of TopK controls the number of non-zero elements in vector g and thus the number of hE_i calculations needed. If one increases N while keeping K fixed, one can increase the model's parameter count while keeping its computational cost effectively constant. Another benefit of Sparse MoE is that Expert output calculations can be parallelized thus allowing to maintain the original model performance even for higher K values.

3.2. Low-Rank Adaptation

Low-rank adaptation or LoRA [6] is a model-training approach that alleviates the training memory requirements by learning a low-rank approximations of gradient updates of feed-forward modules while keeping the original modules unchanged. LoRA has successfully demonstrated its efficiency for both Transformer fine-tuning [8] and pre-training [7] settings, achieving a comparable results to full model training while requiring much less hardware memory.

Formally, let $W_o \in \mathbb{R}^{d_{in} \times d_{out}}$ be the original parameter matrix of feed-forward layer and $\Delta W \in \mathbb{R}^{d_{in} \times d_{out}}$ its corresponding cumulative gradient update after training. Let $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{r \times d_{out}}$ be the low-rank decomposition of ΔW . Then feed-forward layer training process can be represented as:

$$W_o + \Delta W_o = W_o + AB. \quad (2)$$

Consequently, the forward pass of LoRA feed-forward:

$$f(h) = hW_o + \lambda hAB + b_o \quad (3)$$

where $\lambda = \frac{\alpha}{r}$, $\alpha \in \mathbb{N}$ is LoRA scaling factor.

During training W_o is frozen and gradient descent doesn't affect it while AB is optimized separately to approximate the cumulative update addition ΔW . This cuts the necessary memory for optimizer states by factor $\frac{d_{in} \cdot d_{out}}{r \cdot (d_{in} + d_{out})}$ thus increasing the maximal on-device batch size. During inference AB component can be merged with W_o to reduce the computation overhead related to calling separate feed-forward computation kernels.

3.3. Unsupervised dynamic LoRA mixtures (UDLM)

While the original LoRA implementation considers only one adapter (A, B) the method could be generalized to multi-adapter case. Assume we have N adapters $\{(A_i, B_i)\}_{i=1}^N$ and $AB = \sum_{i=1}^N w_i A_i B_i$ for arbitrary vector $w \in \mathbb{R}^N$. Then LoRA feed-forward formula can be expanded:

$$f(h) = hW_o + b_o + \lambda \sum_{i=1}^N w_i hA_i B_i \quad (4)$$

Since the weight vector w is fixed the adapter ensemble can be merged prior network inference which would replicate the Model Soup approach.

To utilize the dynamic ensembling methods it would be enough to define the layer input weight mapping $w : \mathbb{R}^{d_{in}} \mapsto \mathbb{R}^N$. Let $E_i = A_i B_i$ and $w = G(h)$. If $w_i \geq 0$ and $\|w\| = 1$ then the following is equivalent:

$$\lambda \sum_{i=1}^N w_i h_i A_i B_i = \lambda \sum_{i=1}^N G(h)_i h_i E_i = \lambda MoE(h) \quad (5)$$

From which it follows:

$$f(h) = hW_o + b_o + \lambda MoE(h) \quad (6)$$

Thus LoRA dynamic ensembling can be represented through Mixture-of-Experts.

The main issue with MoE-based ensemble is that MoE was designed to be fine-tuned on the training dataset and while we can utilize the pre-trained adapters as trained expert matrices E_i we would still need to tune the router matrix W_G . Previous works[17, 18] explored a straightforward approach of fine-tuning the router matrix on a downstream task using gradient-based methods. However those methods require a substantial amount of labeled data for convergence thus making the approach infeasible in low-resource environments.

An alternative way is to leverage layer spatial statistics for router matrix initialization. Suppose columns of router matrix satisfy $W_G^i = \frac{\hat{W}_G^i}{\|\hat{W}_G^i\|}$ and for any layer input row $h_j = \frac{\hat{h}_j}{\|\hat{h}_j\|}$. Then MoE logit function is equivalent to:

$$MoE\text{-logit}(h) = hW_G = \left[\frac{\hat{h}_j \cdot \hat{W}_G^i}{\|\hat{h}_j\| \|\hat{W}_G^i\|} \right]_{ji} = [\text{cosine-similarity}(\hat{h}_j, \hat{W}_G^i)]_{ji} \quad (7)$$

Hence MoE routing may be reduced to the task of finding the most semantically similar expert.

Expert semantic vector $\hat{W}_G^i$ must describe the cases where the expert is the most efficient. For the set of independently trained LoRA adapters this cases are identified by their corresponding downstream tasks. By aggregating the observed hidden states h during forward pass of neural network on downstream task input we can find the centroid of the task cluster in layer input space. Given the batch of k hidden states $H = \{h^0, h^1, \dots, h^k\}$ the task cluster centroid vector $\hat{W}_G^i$ can be calculated as:

$$\hat{W}_G^i = AGG_{batch}(AGG_{seq}(H)); \quad (8)$$

where $AGG_{seq} : \mathbb{R}^{k \times m \times d_{in}} \mapsto \mathbb{R}^{k \times d_{in}}$ and $AGG_{batch} : \mathbb{R}^{k \times d_{in}} \mapsto \mathbb{R}^{d_{in}}$

3.3.1. Batch aggregation strategies We consider 4 batch aggregation strategies:

Average Simple average of batch elements:

$$AGG_{batch}(X) = \frac{1}{n} \sum_{i=1}^n X_i \quad (9)$$

where $X \in \mathbb{R}^{k \times d_{in}}$.

PCA Principal component analysis (PCA) is a method of dimension reduction that aims to preserve the maximal variance of the original data. By limiting PCA process to the first component $T_1 = XV_1$ we collapse the space X to a single vector which is equivalent to weighted average of features. However, by applying PCA to untransposed batch matrix X the first component weight vector $V_1 \in \mathbb{R}^{d_{in} \times 1}$ would capture the overall feature directions and since W_1 is a unit vector (by PCA design) it is equivalent to finding a centroid in terms of cosine similarity [21].

PCAT By applying PCA dimension reduction to transposed X^T the resulting first component projection $T_1 = X^T V_1$ will correspond to weighted average of rows. The weights in that case would describe contextual semantic diversity thus favoring batch examples with highest network propagation signal magnitude.

PCAM The basis of PCA implementation is singular value decomposition (SVD) of data matrix $X = U\Sigma V^T$ where columns of $V \in \mathbb{R}^{d_{in}, d_{in}}$ correspond to principal directions. SVD only guarantees the uniqueness of singular value matrix $\Sigma \in \mathbb{R}^{k, d_{in}}$ while singular vectors U, V can have multiple valid candidates. Full SVD algorithms are usually deterministic, however there are reduced versions like Truncated SVD and Compact SVD that consider only t most significant singular values of $\Sigma_p = [\sigma_1, \sigma_2, \dots, \sigma_t]$ which can be approximated by exploring the sub-space of data X . Randomized PCA leverages that idea by considering random $X_t \in \mathbb{R}^{k \times t}$ subspace of X for Truncated SVD. Hence we can produce several Randomized PCA solutions $\tilde{V} = V^0, V^1, \dots, V^p$ and then select one with the highest discrimination efficiency between positive and negative examples $pos, neg \in \mathbb{R}^{1 \times d_{in}}$ (layer inputs):

$$PCAM(x) = \arg \max_{V^i \in \tilde{V}} posV^i - negV^i \quad (10)$$

pos, neg examples can be sampled from downstream task input dataset in accordance to networks conditional generation probability.

3.3.2. First layer only router initialization First layer only (FLO) utilizes the fact that Transformer network propagates every layer input through residual connections. If we omit the layer normalization operation the final output of Transformer network T can be approximated as:

$$T(X) = Emb(X) + L_N(Emb(X) + L_{N-1}(Emb(X) + L_{N-2}(Emb(X) + \dots))); \quad (11)$$

where $Emb(X)$ is the embedding matrix that maps the raw input $X \in \mathbb{N}^{m \times 1}$ to corresponding token vectors $E \in \mathbb{R}^{m \times d_{emb}}$ and $L_i(\cdot)$ is the i -th Transformer layer of the network. Thus all intermediate layer inputs of Transformer network lie within the embeddings E space. For the batch of k downstream inputs $\tilde{X} \in \mathbb{R}^{k \times m \times 1}$ the FLO router initialization is defined:

$$\hat{W}_G^i = FLO(X) = AGG_{batch} \left(\frac{1}{m} \sum_{i=1}^m \tilde{E}_{*i*} \right), \quad (12)$$

$$\tilde{E} = Emb(\tilde{X}). \quad (13)$$

For FLO we don't consider AGG_{seq} functions other the sequence-wise average since the first embedding layer is context agnostic (e.g. *BOS* and *EOS* tokens always have the same embedding vector) and as the consequence doesn't exhibit any positional signal bias that could be used in weighed aggregation schemes.

3.3.3. Inter layer embedding router initialization Inter layer embedding (ILE) initialization uses the respective intermediate layer outputs $L_l(\cdot)$ to calculate layer $l+1$ exact input statistics. While obtaining all layer outputs is more computationally expensive than taking the outputs of embedding layer only this process also yields the logits of output tokens and thus the probabilities $P(X)$ of the input sequence X (without target sequence the Transformer network treats the next input token as the model output).

Let $L_{<l}(X)$ be the output of layer L_{l-1} produced for model input X . Then for the batch of k downstream inputs $\tilde{X} \in \mathbb{R}^{k \times m \times 1}$ the router initialization of network layer l is defined as:

$$\hat{W}_G^i = ILE(X) = AGG_{batch} \left(Conf(\tilde{X}) \cdot AGG_{seq}(L_{<l}(X)) \right), \quad (14)$$

$$Conf(\tilde{X}) = \left(1 + \beta \frac{P^*(\tilde{X}) - \min_j P(X_{j**})}{\max_j P(X_{j**})} \right) \quad (15)$$

$$P^*(\tilde{X}) = (P(X_{1**}), P(X_{2**}), \dots, P(X_{k**})) \quad (16)$$

where $\beta \geq 0$ is oversampling coefficient. If $\beta = 1$ then the signal of most confident input is duplicated and hence the batch statistics will be shifted towards its semantics.

Unlike embedding layer, Transformer layer outputs reflect the context (input is mixed by attention module) and it may be beneficial to bias sequence aggregation function AGG_{seq} towards

service tokens such as *BOS* and *EOS*. Therefore in ILE initialization we consider three options for AGG_{seq} : average of sequence hidden states, *BOS* token hidden state, *EOS* token hidden state.

4. EXPERIMENTAL SETUP

Following previous works [8, 11, 17] on LoRA ensembles we measure the efficiency of our method for both knowledge transfer and knowledge boosting settings. For the task we chose abstractive summarization as it has a wide variety of datasets of different domains.

4.1. Datasets

To evaluate summarization performance we consider 6 datasets:

CNN / Daily Mail [23]. Originally the CNN / Daily Mail news dataset was designed for question answering task but was later repurposed by Nallapati et al. [23] for abstractive summarization. Each news story is supplied with a set of human-written highlight sentences that serve as a summarization target. Number of examples: 312k.

Xsum [26]. A collection of articles from BBC news portal created for extremely condensed summarization task. While the summaries are more abstractive than of CNN / Daily Mail they are also more noisy due to fully automated HTML parsing process that didn't include summary-source semantic connection verification. Number of examples: 226k.

Wikihow [22]. A large-scale dataset of instructions from the online WikiHow.com website. Each instruction article is presented in form of multiple paragraphs, each describing the details of the next step. The summaries were obtained by concatenating the titles (highlights) of each instruction step paragraph. Number of examples: 200k.

Booksum [24]. A collection of datasets for long-form narrative summarization of texts from the literature domain. Booksum contains human-written summaries on three levels of granularity: paragraph, chapter, and book. Due to input-length limitations of summarization models, in this work, we consider only paragraph level. Number of examples: 146k.

SAMSum [25]. A corpus of messenger dialogue conversations with human-composed summaries. Both conversations and summaries were written by linguistic experts and then additionally cross-validated by the same experts to correct possible semantic, structure or typing errors. As a result Samsum examples are of highest quality and hence are often used for summarization model validation. Number of examples: 16k.

SciTLDR [27]. A multi-reference science article summarization dataset collected from OpenReview portal. The summaries were obtained by parsing the author-written article TLDR section and by rewriting peer-reviews. All summaries were assessed by experts and inappropriate summaries were fully discarded. To address the model input limitations the dataset provides 3 variants of input document: abstract-only; a concatenation of Abstract, Introduction and Conclusion sections (AIC) and the full article text. For experiments we consider only the official author-provided TLDR summary and AIC variant of the input document. Number of examples: 3.2K.

4.2. Model and baselines

As the base model for our experiments we employ BART [28] large, that was shown [29] to achieve state-of-the-art results on considered datasets in both zero-shot and fine-tuning settings.

For the baselines we consider the following static ensembling configurations: **LoRA** adapter tuned on the target dataset; **Average merge** of all trained LoRA adapters; the best weighted merge of trained adapters according to **LoraHub**[8] algorithm for target data (**data-tuned**) and **universal** (tuned on examples from every dataset).

For UDLM ensembles beside different initializations and expert counts we consider two expert mapping levels: **tokens** and **layer**. In case of layer level the layer input is first aggregated using the same sequence aggregation function as in router initialization and then passed as a single-token input to MoE router layer the result of which is mapped to all tokens of the original layer input sequence.

Table 1. Data-wise LoRA tuning performance for different decomposition ranks.

Rank	CNN / DM			SAMSum		
	R-1	R-2	R-L	R-1	R-2	R-L
16	41.47	19.29	28.14	51.27	26.91	42.18
32	41.77	19.66	28.68	51.88	27.39	42.71
64	42.11	19.97	29.14	52.41	28.08	43.49
128	42.04	19.80	29.13	52.17	27.99	43.40
256	41.93	19.82	29.07	52.06	28.12	43.31

Table 2. LoRA tuning results and relative ROUGE improvement over BART zero-shot generation.

Data	R-1 _{LoRA}	R-2 _{LoRA}	R-L _{LoRA}	$\frac{R-1_{LoRA}}{R-1_{zero}}$	$\frac{R-2_{LoRA}}{R-2_{zero}}$	$\frac{R-L_{LoRA}}{R-L_{zero}}$
CNN / DM	43.08	20.75	30.03	1.09	1.17	1.23
Xsum	44.19	21.02	35.88	2.28	7.66	2.52
SAMSum	52.34	28.05	43.50	1.65	3.34	1.68
Booksum	21.17	4.57	16.50	1.20	1.23	1.21
Wikihow	43.92	19.11	34.22	1.48	2.65	1.79
SciTLDR	26.64	11.13	20.13	1.15	1.39	1.20

4.3. Implementation details

LoRA adapters were tuned using Adam optimizer with learning rate $lr = 1e - 4$, linear scheduler with warmup for 5 epoch and early stopping with evaluation on validation part every 0.5 epoch. To mimic the full tuning we trained all feed-forward block of Transformer encoder and decoder layers which is W_k, W_v, W_q projection matrices of attention mechanism, W_o multi-head attention aggregation matrix and W_{down}, W_{up} of post-attention MLP. The rank r of LoRA decompositions was determined by our pilot experiments (see Table 1) on CNN / Daily Mail and SAMSum datasets where we found $r = 64$ and $\alpha = 2r$ to have highest average performance. The router adapters and LoraHub ensembles were initialized on the same subset of 100 examples of training split from each dataset. To facilitate comparison we provide relative ROUGE improvement (for 1, 2 and L modes) over the data-tuned LoRA (see Table 2 for data-wise $\mathbf{R-X}_{LoRA}$ scores):

$$\Delta \mathbf{R-X} = \frac{\mathbf{R-X}}{\mathbf{R-X}_{LoRA}} - 1 \quad (17)$$

where $\mathbf{R-X} = \mathbf{ROUGE-X}$ and $\mathbf{X} \in \{1, 2, L\}$.

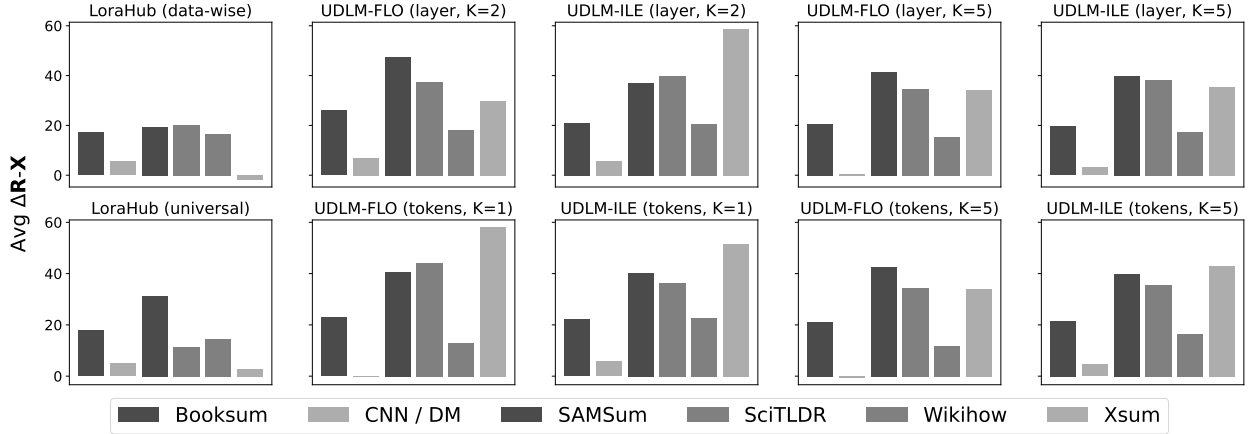
5. EXPERIMENT RESULTS

5.1. Knowledge Transfer

Table 3 reports the evaluation results for knowledge transfer setting (i.e. $\mathbf{R-X}_{LoRA} = \mathbf{R-X}_{zero}$). As expected all ensembling methods show positive improvement over the zero-shot inference, however UDLM exhibits substantially higher efficiency with up-to 30% average relative ROUGE improvement. While the performance difference between best FLO and ILE *TopK* filtered configurations is marginal the ILE initialization has less performance drop for full adapter mixture (*TopK* = 5) implying that the method has better data-adapter alignment accuracy. This fact is reflected in Figure 1 which shows the data-wise improvements. Both FLO and ILE initialization captures the LoRA tuning trends (see Table 2, right column block) having the highest improvements for Xsum, SAMSum, although, despite the same adapter domain, FLO struggles to apply Xsum knowledge for CNN / Daily Mail summarization and significantly underperforms on Wikihow.

Table 3. Best knowledge transfer results for each method by average ROUGE improvement.

Method	AGG_{batch}	AGG_{seq}	β	Mapping	TopK	$\Delta R-1$	$\Delta R-2$	$\Delta R-L$	AVG
Zero-shot	-	-	-	-	-	0.00	0.00	0.00	0.00
Average merge	-	-	-	-	-	5.23	11.37	6.34	7.65
LoraHub (data-tuned)	-	-	-	-	-	8.22	15.49	14.43	12.71
LoraHub (universal)	-	-	-	-	-	9.87	16.55	14.48	13.63
UDLM-FLO	mean	mean	-	layer	2	18.43	39.96	24.16	27.51
	mean	mean	-	tokens	1	18.60	43.76	26.58	29.65
	pcaT	mean	-	layer	5	16.59	32.62	23.53	24.25
	mean	mean	-	tokens	5	16.15	32.82	22.37	23.78
UDLM-ILE	pcaM	mean	0.0	layer	2	18.86	44.93	27.02	30.27
	pca	<i>EOS</i>	1.0	tokens	1	19.94	41.53	27.37	29.62
	pcaM	<i>BOS</i>	1.0	layer	5	18.00	33.86	24.43	25.43
	pcaM	mean	1.0	tokens	5	18.49	35.99	25.49	26.65

**Figure 1.** Data-wise relative improvements of knowledge transfer configurations.

5.2. Knowledge Boosting

Table 4 show the results of best knowledge boosting configurations. Unlike knowledge transfer where model has no experience, in this setting any added knowledge may conflict with established task-solving patterns of the data-tuned adapter. For that reason Average merge leads to guaranteed performance degradation and even the best LoraHub data-tuned configurations struggle to maintain the positive effect for all datasets (see Figure 2). Similar situation is observe for UDLM-FLO that performs worse than data-tuned LoraHub ensembles yet better than universal mixture. UDLM-ILE is the only method to achieve positive average relative ROUGE improvements.

From Figure 2 we can see that methods are generally the least efficient for datasets with highest LoRA ROUGE improvements over the zero-shot (see Table 2, right column block) and only UDLM-ILE manage to introduce new data insights. This indicates that without embedding tuning LoRA adapters tend to shift inter-layer embeddings from the original values with higher magnitude to compensate the data-model vocabulary misalignment (e.g. homonymy resolution or different punctuation habits). ILE being a layer-based method alleviates the issue with layer-specific statistics that account for those hidden embedding manipulations, however, it still introduces performance degradation for SAMSum as it's documents has the most distinct grammatical features

such as ascii emojis (i.e. irregular punctuation usage) abbreviations (e.g. imho) and extremely short sentences (natural for messenger dialogues).

Table 4. Best method knowledge boosting results by average ROUGE improvement.

Method	AGG_{batch}	AGG_{seq}	β	Mapping	TopK	$\Delta R-1$	$\Delta R-2$	$\Delta R-L$	AVG
LoRA	-	-	-	-	-	0.00	0.00	0.00	0.00
Average merge	-	-	-	-	-	-19.91	-35.62	-24.55	-26.69
LoraHub (data-tuned)	-	-	-	-	-	-2.98	-5.04	-0.54	-2.85
LoraHub (universal)	-	-	-	-	-	-15.51	-30.66	-17.80	-21.32
UDLM-FLO	pcaT	mean	-	layer	1	-8.37	-17.59	-7.58	-11.18
	mean	mean	-	tokens	2	-7.77	-15.68	-7.68	-10.37
	pcaT	mean	-	layer	6	-8.26	-17.16	-8.72	-11.38
	pcaT	mean	-	tokens	6	-8.81	-18.03	-9.33	-12.06
UDLM-ILE	pcaM	<i>BOS</i>	1.0	layer	1	4.62	5.21	6.37	5.40
	pcaT	<i>BOS</i>	0.0	tokens	2	2.68	3.28	3.60	3.19
	pcaM	<i>EOS</i>	0.0	layer	6	4.32	2.68	3.94	3.65
	pcaM	mean	0.0	tokens	6	4.19	1.99	4.28	3.49

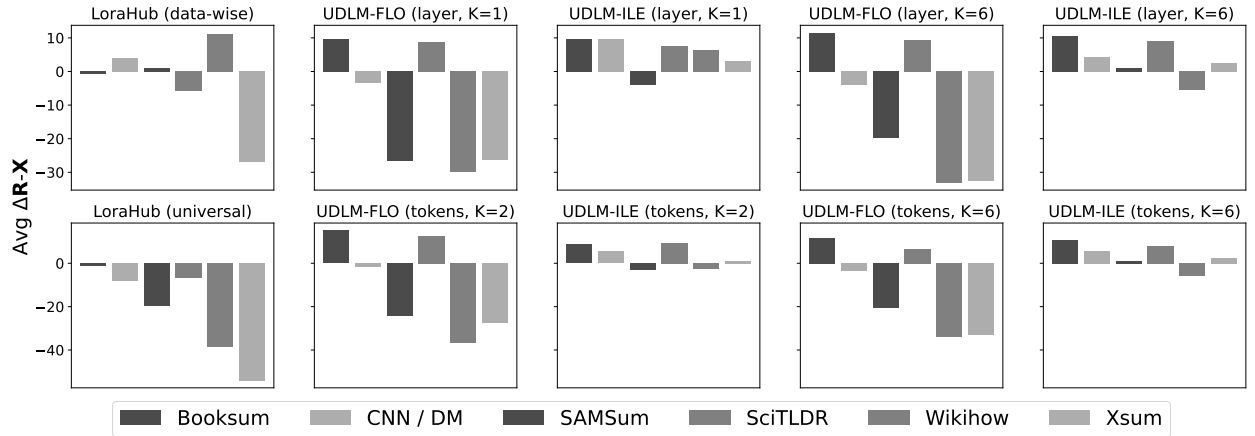


Figure 2. Data-wise improvements of best knowledge boosting configurations.

5.3. UDLM-ILE parameter effects

Evaluation determined that UDLM-ILE with layer level input mapping is the most robust dynamic ensembling method. However the performance difference between different ILE configurations can be considered marginal in some cases which suggests an existence of robust hyperparameter set for both knowledge transfer (KT) and knowledge boosting (KB).

Figure 3 shows the best average relative ROUGE improvement for each parameter setting (for comparison reasons we calculate the metric as if $R-X_{LoRA} = R-X_{zero}$). First most noticeable trait is that $AGG_{batch} = \text{pcaM}$ batch aggregation method remains the best for all configurations thus eliminating the need for AGG_{batch} tuning. Sequence aggregation is less straightforward. While *BOS* aggregation significantly outperforms other strategies during knowledge boosting it is less

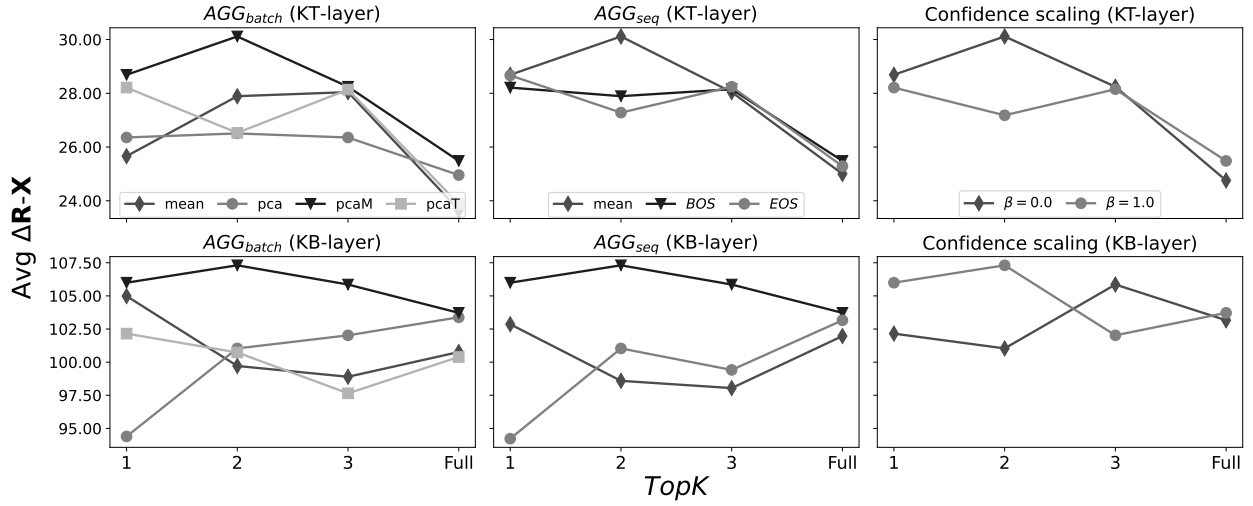


Figure 3. Best results for each *ILE* parameter with respect to adapter mixture mode and *TopK*.

efficient for knowledge transfer where mean aggregation achieves the best improvements. On the other hand the performance difference between sequence aggregation methods during knowledge transfer is marginal for all cases but $TopK = 2$ which implies that current observation is likely to be the outlier and using $AGG_{seq} = BOS$ for both settings could be optimal.

The confidence scaling has different effect depending on $TopK$ and adapter mixture mode. For knowledge transfer keeping $\beta = 0$ leads to better performance but at the same time $\beta = 1$ can significantly enhance the effectiveness of UDLM-ILE knowledge boosting. Since confidence scaling utilizes the pre-trained adapters to calculate model input token sequence output probability this implies that $\beta \geq 0$ improves the router embedding dissimilarity, biasing the expert weights towards semantically closest adapters. In case of knowledge transfer it is inefficient as adapters were tuned for different domains and summary structure, however for knowledge boosting it improves the utilization of the data-tuned expert. Therefore the optimal setting for β is highly reliant on the adapter set.

Table 5. Performance benchmark of model forward-pass on SAMSum dataset.

	Base	LoRA	UDLM-ILE	UDLM-ILE	MoE	MoE
	(unmerged)		(6 experts, layer)	(6 experts, tokens)	(6 experts, layer)	(6 experts, tokens)
GPU Memory (MB)	3622	4010	4806	4806	12568	12568
Memory factor	1	1.11	1.33	1.33	3.47	3.47
GPU Time (ms)	70	101	137	163	227	321
Speed factor	1	0.69	0.51	0.43	0.31	0.22

6. COMPUTATIONAL PERFORMANCE IMPACT

An important part of any architectural optimization is the computational feasibility for practical applications. UDLM relies on LoRA adapters that despite lower memory footprint have higher CUDA kernel launching overhead due to 2 additional feed-forward matrix operations. Unlike the expert count, this overhead cannot be negated by device parallelization. At the same time UDLM can't be merged with the base model like the regular LoRA or static LoRA ensembles.

To measure the performance impact we benchmarked (see Table 5) memory and time difference for forward pass on Nvidia RTX 4090 for the batch of 10 examples of 512 tokens from SAMSum

dataset. Regular LoRA and LoRA static ensembles (i.e. weighted sum of adapter weights) with decomposition rank 64 reduce the computational speed by 31% while increasing the peak memory usage by 11%. Although UDLM introduces additional router feed-forward matrices the performance impact for 6 equivalently-sized experts with layer-wise mapping is comparable to LoRA with just additional 22% memory usage and 35% more latency. A corresponding full MoE configuration has more drastic performance drop exhibiting a tripled latency and 3.5 times increased memory requirements. Since token-wise mapping only improves the expert activation diversity it has the same memory footprint but requires more expert feed-forward calculations thus reducing the throughput by 57% and 78% for UDLM and full MoE respectively.

Therefore UDLM has comparable memory costs to LoRA static ensembles and almost 3 times more memory efficient than MoE. The latency gap with unmerged LoRA can be closed by applying the expert adapters concurrently, however at the cost of substantially higher peak memory usage per layer which for instance considering the largest BART matrix $W_{up} \in \mathbb{R}^{1024 \times 4096}$ and current setting would result in additional $5 \cdot \frac{32 \cdot 10 \cdot 512 \cdot 4096}{1024^2} = 3200\text{MBs}$ peak GPU memory usage.

7. CONCLUSION

In this work we proposed Unsupervised Dynamic LoRA Mixtures (UDLM), a dynamic adapter ensembling approach that unifies the pre-trained LoRA adapters into Mixture-of-Expert (MoE) blocks using layer-wise inference statistics for unsupervised router matrix initialization. We explored two variants of UDLM statistic accumulation for Transformer networks: First-layer only (FLO) that initializes all router matrices with the same token embedding layer expert centroids and Inter-layer embeddings (ILE) that leverages respective layer input hidden states for layer-wise matrix initialization. The approach was tested for knowledge transfer and knowledge boosting settings on 6 abstractive summarization datasets of various domains. For knowledge transfer both UDLM variants demonstrated a threefold improvement over the best static ensembling method, LoraHub, however, in knowledge boosting mode only UDLM-ILE managed to boost the performance of data-tuned LoRA adapters by a significant margin while other methods degraded the summarization quality. The best UDLM-ILE configuration with layer level expert routing also proved to be the most computational efficient, having only 20% increased memory usage and 35 % increased latency over static LoRA ensembles.

Funding. The work of Daniil Chernyshev was supported by Non-commercial Foundation for Support of Science and Education "INTELLECT".

REFERENCES

1. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts", in *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.
2. A. Q. Jiang, et al. "Mixtral of experts.", arXiv:2401.04088, 2024.
3. B. Lin, et al., "MoE-LLaVA: Mixture of Experts for Large Vision-Language Models.", arXiv:2401.15947, 2024.
4. N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer ", *ICLR 2017*, 2016.
5. W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning ", arXiv:2209.01667, 2022.
6. E. J. Hu, et al., "LoRA: Low-Rank Adaptation of Large Language Models. ", *ICLR 2022*, 2021.
7. V. Lialin, S. Muckatira, N. Shivagunde, and A. Rumshisky, "ReLoRA: High-Rank Training Through Low-Rank Updates", In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.
8. C. Huang, Q. Liu, B. Y. Lin, C. Du, T. Pang, and M. Lin, "LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition", arXiv:2307.13269, 2023.
9. Y. Wang, S. Agarwal, S. Mukherjee, X. Liu, J. Gao, A. Hassan, and J. Gao, "AdaMix: Mixture-of-Adaptations for Parameter-efficient Model Tuning", In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5744–5760, 2022.
10. Sheng S., et al., "Mixture-of-Experts Meets Instruction Tuning: A Winning Combination for Large Language Models", In *The Twelfth International Conference on Learning Representations*, 2024.
11. A. Chronopoulou, M. E. Peters, A. Fraser, and J. Dodge, "AdapterSoup: Weight Averaging to Improve Generalization of Pretrained Language Models", *EACL 2023*, pp. 2009-2018, 2023.
12. M. Wortsman, et al., "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time", *ICML 2022*, pp. 23965-23998, 2022.

13. X. Lv, et al. "Parameter-efficient Weight Ensembling Facilitates Task-level Knowledge Transfer", In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 270–282, 2023.
14. G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi, "Editing models with task arithmetic", In The Eleventh International Conference on Learning Representations, 2022.
15. J. Zhang, J. Liu, and J. He, "Composing Parameter-Efficient Modules with Arithmetic Operation", Advances in Neural Information Processing Systems, 36, 2024.
16. A. Chronopoulou, J. Pfeiffer, J. Maynez, X. Wang, S. Ruder, and P. Agrawal, "Language and Task Arithmetic with Parameter-Efficient Layers for Zero-Shot Summarization", arXiv:2311.09344, 2023.
17. E. M. Ponti, A. Sordoni, Y. Bengio, and S. Reddy, "Combining Parameter-efficient Modules for Task-level Generalisation", EACL 2023, pp. 687–702, 2023.
18. S. Dou, et al., "Loramoe: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment", arXiv:2312.09979, 2023.
19. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners", OpenAI blog, 1(8), 9, 2019.
20. Y. Liu, et al., "Roberta: A robustly optimized bert pretraining approach", arXiv:1907.11692, 2019.
21. A. Zou, et al., "Representation engineering: A top-down approach to ai transparency", arXiv:2310.01405, 2023.
22. M. Koupaei and W. Y. Wang, "WikiHow: A Large, Scale Text Summarization Dataset", arXiv:1810.09305, 2018.
23. R. Nallapati, B. Zhou, C. d. Santos, C. Gulcehre, and B. Xiang, "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond", Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, pp. 280–290, 2016.
24. W. Kryscinski, N. Rajani, D. Agarwal, C. Xiong, and D. Radev, "BookSum: A Collection of Datasets for Long-form Narrative Summarization", arXiv:2105.08209, 2021.
25. B. Gliwa, I. Mochol, M. Biesek, and A. Wawer, "SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization", in Proceedings of the 2nd Workshop on New Frontiers in Summarization, pp. 70–79, 2019.
26. S. Narayan, S. Cohen, and M. Lapata, "Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization", EMNLP 2018, pp. 1797–1807, 2018.
27. I. Cachola, K. Lo, A. Cohan, and D. S. Weld, "TLDR: Extreme Summarization of Scientific Documents", In Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 4766–4777, 2020.
28. M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension", Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7871–7880, 2020.
29. Y. Chen, et al, "UniSumm and SummZoo: Unified Model and Diverse Benchmark for Few-Shot Summarization ", In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 12833–12855, 2023.
30. Q. Liu, et al. "Moelora: An moe-based parameter efficient fine-tuning method for multi-task medical applications. ", arXiv:2310.18339, 2023.