Parameter-Efficient Methods

for Large Language Models

Xaver Maria Krückl

Ludwig-Maximilians-Universität München

Masterseminar 'The GPTs and Their Ilk' @CIS, SoSe 2023

Prof. Hinrich Schütze, Haotian Ye, M.Sc.

19.05.2023

Overview

Introduction

- many NLP applications rely on large-scale pre-trained language models (LLMs)
- often, only **one** model is adapted to **multiple** downstream tasks
- but: powerful LLMs are huge and adapting them via fine-tuning is problematic
- tuning all original model parameters is computationally, financially and environmentally costly

→ only certain industrial labs can afford to develop and do research on the largest models

→ such models cannot be widely deployed on everyday devices like regular computers or mobile phones

→ exemplary, the carbon footprint of training BLOOM (176B parameters) is up to 50 tonnes of CO_2eq (Luccioni et al., 2022)

Motivation

- necessity of methods that make training and using LLMs more parameter-efficient

→ (preferably) without changing many parameters and a decrease in performance

→ solving the previously presented problems

- different approaches building on a handful of basic methods:
  - additive: **AdapterHub** (Adapters)
  - selective: **BitFit**
  - based on reparametrization: **LoRA**

(Lialin et al., 2023), (Hu et al., 2021), (Pfeiffer et al., 2020b)

AdapterHub - Overview

- a framework to dynamically integrate pre-trained adapters for different NLP tasks and languages
- includes all recent Adapter frameworks, e.g. by Houlsby et al. (2019) or Pfeiffer et al. (2020a)
- built on top of HuggingFace's Transformers library:

→ easily adaptable to soa pre-trained models (e.g. BERT, RoBERTa, XLM-R), only requiring 2 additional lines of code

→ open-source framework, automatically extracts and stores adapter weights

→ allows to easily download, train and share task-specific adapters and models

(Pfeiffer et al., 2020b)

Adapters - Basics

- neural modules introducing a small set of new parameters Φ_l at every Transformer layer l
- Φ are learnt on a target task while keeping the pre-trained parameters Θ frozen/fixed

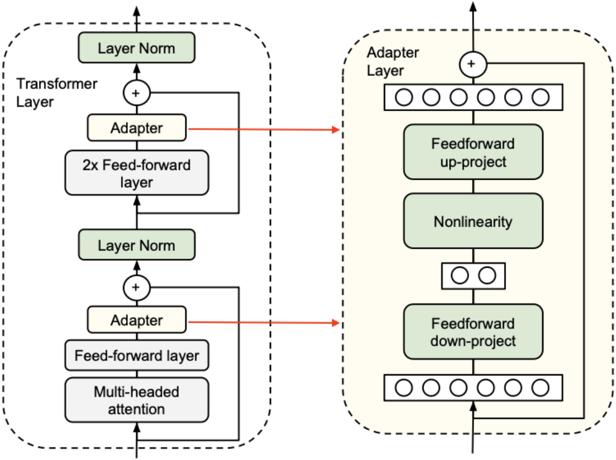
$\rightarrow$ Φ learn to encode task-specific representations in intermediate layers of the pre-trained model

- empirical validation: two-layer feed-forward neural network with bottleneck works well (Houlsby et al., 2019)
- how to address multiple tasks?

$\rightarrow$ training many task- and language-specific adapters for the same model $\rightarrow$ can be exchanged and combined post-hoc $\rightarrow$ **AdapterHub** $\rightarrow$ efficient parameter sharing $\rightarrow$ strong results in multi-task and cross-lingual transfer learning

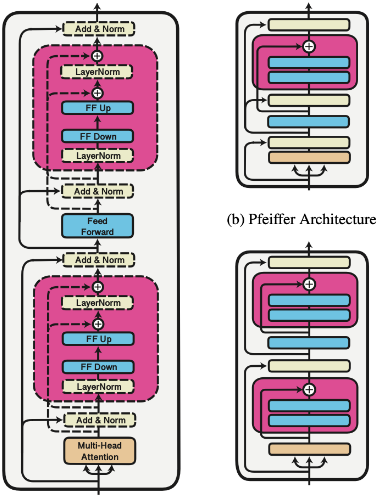
(Pfeiffer et al., 2020b)

Adapters - Structure



Houlsby et al. (2019), p. 3

AdapterHub - Structure



(a) Configuration Possibilities

(b) Pfeiffer Architecture

(c) Houlsby Architecture

Pfeiffer et al. (2020b), p. 5

AdapterHub - Advantages

- Adapters are small, scalable and shareable:

$\rightarrow$ $>99\%$ of parameters required for target tasks fixed during training $\rightarrow$ can be shared across models

$\rightarrow$ **2** fully fine-tuned Bert-Base models require the same storage space as **125** tuned with adapters (440Mb)

- modularity of representations:

$\rightarrow$ frozen parameters Θ force adapters to learn output representation compatible with subsequent transformer layers

$\rightarrow$ possibility of replacing adapters dynamically or stacking them on top of each other

- non-interfering composition of information:

$\rightarrow$ encapsulation forces adapters to learn representations compatible across tasks

$\rightarrow$ reducing issues with catastrophic forgetting and interference

Adapters/AdapterHub - Limitations

- sensitivity on size and placement of adapters within transformer blocks
- how/should new normalization layers be introduced?
- extending the model depth by using adapters increases inference time or latency
- AdapterHub depends on available adapters for specific tasks and languages
- quality of existing adapters may vary

(Pfeiffer et al., 2020b), (Hu et al., 2021)

BitFit - Overview

- BitFit = **BIas-Term FIne-Tuning**
- training only the bias-terms and task-specific classification layers
- rest of the transformer's encoder parameters Θ are frozen
- bias terms are additive - only correspond to a very small fraction of the model parameters
- BitFit is effective for pre-trained BERT models:

$\rightarrow$ bias parameters only make up 0.09/0.08% of all parameters in $\text{BERT}_{\text{BASE/LARGE}}$

$\rightarrow$ efficient to store only parameter vectors of bias terms and final classifier layer for each new task (0.1% of all parameters)

- fine-tuning viewed as *exposing knowledge induced by training*, not learning new, task-specific linguistic knowledge

(Ben-Zaken et al., 2022), (Hu et al., 2021)

BitFit - Background I

- in a BERT encoder of L layers, each layer l starts with M self-attention heads
- a self attention head (m, l) has *key*, *query* and *value* encoders in the form of a linear layer:
  (x is the output of the former encoder, initially of the embedding layer):

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

  $\rightarrow$ combined using an attention mechanism **not** involving new parameters:

$$\mathbf{h}_1^\ell = att\big(\mathbf{Q}^{1,\ell}, \mathbf{K}^{1,\ell}, \mathbf{V}^{1,\ell}, .., \mathbf{Q}^{m,\ell}, \mathbf{K}^{m,\ell}, \mathbf{V}^{m,l}\big)$$

(Ben-Zaken et al., 2022), p. 2

BitFit - Background II

- injected into a MLP with layer-norm LN:

  $\rightarrow$

$$
\begin{aligned}
\mathbf{h}_2^\ell &= \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right) \\
\mathbf{h}_3^\ell &= \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell \\
\mathbf{h}_4^\ell &= \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right) \\
\mathbf{h}_5^\ell &= \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right) \\
\mathbf{out}^\ell &= \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell
\end{aligned}
$$

- matrices and vectors in blue: Θ, frozen
- red: vectors of bias terms - to be trained
- only fine-tuning b_q and b_{m_2} achieves almost full-model accuracy

(Ben-Zaken et al., 2022), p. 2

BitFit - Advantages/Limitations

Advantages:

- very few parameters per fine-tuned task are changed
- same set of parameters is changed for every task
- changed parameters are isolated and localized across the entire parameter space
- for pre-trained BERT models: sometimes even better than fine-tuning the entire model

Limitation(s):

- method only 'competitive' with other fine-tuning methods for larger models

(Ben-Zaken et al., 2022), (Hu et al., 2021)

LoRA - Overview and Hypothesis

- **Inspiration**: learned, over-parameterized models only build upon a few, *intrinsic* dimensions (Aghajanyan et al., 2020)
- i.e. they can still learn effectively even if randomly projected to a smaller subspace

Hypothesis:

Updates to weights during model adaptation/fine tuning also only have a low intrinsic rank.
→ Pre-trained weight matrices can be represented using a low-rank decomposition!

→ **LoRA**: optimizing *rank decomposition matrices* of some dense layers during adaptation trains these indirectly (when keeping the pre-trained weights frozen)

Hu et al. (2021)

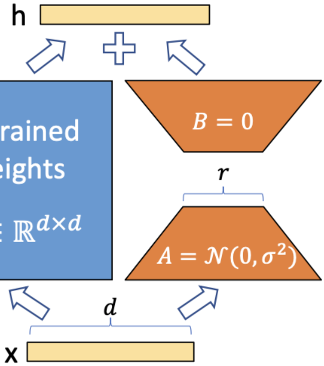
LoRA - Method

- given pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, it's update is constrained with a *low-rank decomposition*:
  $\boldsymbol{W}_0 + \Delta \boldsymbol{W} = \boldsymbol{W}_0 + \boldsymbol{BA}$
  where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$ and the rank $r \ll min(d,\, k)$
- during training, W_0 is frozen (no gradient updates), A and B contain trainable parameters
- weight matrices are multiplied with the same input, output vectors are summed coordinate-wise
- for $h = W_0 x$, the modified forward pass yields
  $\boldsymbol{h} = \boldsymbol{W}_0 \boldsymbol{x} + \Delta \boldsymbol{W} \boldsymbol{x} = \boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{BAx}$
- random Gaussian initialization for A and zero for B, so $\Delta W = BA$ is zero when training starts; then ΔWx is scaled $\frac{\alpha}{r}$
- α is a constant in r, actually the first r being tried

LoRA: Essential Part - Reparametrization

- pretrained weights W_0 are frozen, only A and B are trained
- dimensionality d remains the same, but:

→ much lower rank r significantly reduces computational efforts



Hu et al. (2021), p. 1

LoRA - Generalizations

- general form of fine-tuning: allows to only train a subset of pre-trained parameters
- LoRA: update to weight matrices during training does not need to have full rank (all accumulated gradients)

→ when applying LoRA to all weight matrices and setting r to rank of pre-trained matrices - 'full' fine-tuning is performed

- LoRA applicable to any subset of weight matrices in neural networks

→ adaption for downstream tasks here limited to matrices W_q, W_k, W_v, W_o in the self attention module

→ matrices in the MLP module (2) are frozen (for simplicity and further parameter-efficency)

LoRA - Experiments

- evaluating downstream performance of LoRA on RoBERTa, De-BERTa, GPT-2 and GPT-3 175B
- covering tasks from NLU to NLG
- using common benchmarks (e.g. GLUE, WikiSQL) and data-sets
- replicated baselines setups from other parameter-efficient methods:
  - **Fine-Tuning**
  - Bias-only/**BitFit**
  - Prefix-embedding/Prefix-layer Tuning
  - **Adapter Tuning**

LoRA - Results

Results on GPT-3 175B:

→ LoRA achieved the best validation results using only the fewest trainable parameters on three tasks

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Hu et al. (2021), p. 8

LoRA x GPT-3

To which weight matrices in a Transformer should LoRA be applied?

- adapting both W_q and W_v provided the best results
- results of adapting all attention weights provides equally good results at low rank - but: needs more computation

What is the optimal rank r for LoRA?

	Weight Type	$r = \boxed{1}$	$r = 2$	$r = 4$	$r = \boxed{8}$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Hu et al. (2021), p. 10

$\rightarrow$ validation accuracy of low ranks better than for higher ranks!

LoRA - General Advantages

- simple linear design - allows to merge trainable matrices with frozen weights:

→ no inference latency (vs. Adapters!)

→ freezing pre-trained weights prevents forgetting issues

- possibility of sharing a pre-trained model for several tasks - building small LoRA modules for each:

→ switching tasks only means subtracting matrices A and B from W_0 and replacing them by A' and B'

- no need to calculate gradients and only need to optimize smaller, low rank matrices:

→ more efficient training and reduced storage requirement

→ memory-efficiency allows fine-tuning on consumer GPUs (e.g. Kaggle or Colab notebooks)

Hu et al. (2021)

LoRA - Concrete Advantages

in terms of Parameter-Efficiency:

- for a large Transformer trained with Adam:

→ reduces VRAM usage by up to 2/3 if $r \ll min(d_{model})$

→ reduces GPT-3 175B from 1.2TB to 350GB

→ with $r = 4$ and adaption of W_q and W_v, checkpoint size is reduced by around 10.000× from 350GB to 35MB

→ allows to train with fewer GPUs and avoids I/O bottlenecks

- 25% of speedup during training on GPT-3 175B as compared to full fine-tuning

Hu et al. (2021)

LoRA - Limitations

- when trying to eliminate additional inference latency by absorbing A and B into W:

$\rightarrow$ not straightforward to batch inputs to different tasks with different A and B in a single forward pass

- LoRA only applied to attention head matrices, mostly done by using heuristics

$\rightarrow$ more principled ways to do this?

$\rightarrow$ maybe even better results are possible...

- also no complete clarification by LoRA on how features are learned during pre-training to perform well on downstream tasks (general issue)

Hu et al. (2021)

Conclusion

- considerable advances in parameter efficiency across all presented frameworks
- LoRA addresses flaws of previous approaches regarding inference latency
- LoRA might also only converge to training the original model

→ adapter-based methods converge to an MLP

→ prefix-based methods converge to a model unable to deal with long input sequences

- LoRA appears to be quite influential and is not even at it's final state

→ more development to come

Bibliography I

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, 2020.

Elad Ben-Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.acl-short.1`.

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 Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/houlsby19a.html`.

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, 2021.

Bibliography II

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning, 2023.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *ArXiv*, abs/2211.02001, 2022.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2020a.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pages 46–54, Online, 2020b. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.7`.