Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Chair XII for Natural Language Processing, Universität Würzburg

1. Exercise for "Multilingual Natural Language Processing"

28.06.2024

1 Paper Readings

The PEFT literature is vast and grows rapidly. The papers listed below serve as an initial starting point for your reading to complete the homework.

- Towards A Unified View of Parameter-Efficient Transfer Learning
- MAD-X: An Adapter-Based Framework For Multi-Task Cross-Lingual Transfer
- LoRA: Low-Rank Adaption of Large Language Models
- Prefix-Tuning: Optimizing Continuous Prompts for Generation

2 Parameter-Efficient Fine-Tuning: Basics

1. Describe the core idea of parameter-efficient fine-tuning (PEFT) briefly.

> PEFT refers to a group of fine-tuning techniques in which typically a small fraction ($\leq 5\%$, typically $\leq 1\%$) of existing or newly added parameters are fine-tuned.

2. Concisely explain the key advantages of PEFT!

> Clear advantages of PEFT techniques are:
>
> - **VRAM savings:** modern optimizers (cf. Adam(W)) store copies of trainable parameters to be able to perform second order updates which requires

abundant VRAM if the model is fully fine-tuned – this enables to fine-tune billion parameter-sized models (paired with 8bit precision training) on consumer hardware at almost full fine-tuning performance

- **memory savings via modularity:** if only a fraction of the parameters a trained, we do not need to store entire models per task

Debatable advantages of PEFT techniques are:

- **Faster training:** training speed primarily accelerated by fitting larger batch sizes (due to VRAM savings) and potentially better training stability at larger learning rates
- **Stability:** prior work suggests that PEFT can be more robust to varying hyperparameters

3. Can you think of and explain potential disadvantages oft PEFT?

Practical disadvantages of PEFT are:

- **Performance**: prior work frequently makes it look like PEFT outperforms full fine-tuning. In practice, however, follow-up work hardly ever has been able to exceed full fine-tuning performance (cf. Towards A Unified View of Parameter-Efficient Transfer Learning)
- **Inference**: depending on the PEFT approach, inference latency may be c. $10-30\%$ higher, as input sequence length increases (e.g. prefix-tuning) or the model becomes deeper (e.g. adapters)
- **Technical debt**: PEFT frameworks are wrappers around wrappers (`transformers` library); these libraries typically end-up playing "catch-up" to latest research developments

3 Comparison of methods

Analyse and compare (i) LoRA, (ii) Prefix-Tuning, and (iii) Adapters along the following dimensions:

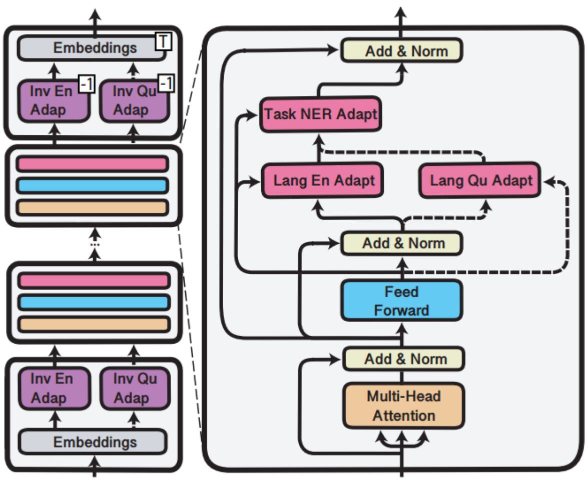
- Modelling: how are the original language model representations updated during PEFT between the approaches?
- Implementation, ease of use

- Inference

Parameter-Efficient Transfer Learning

Adapters: Added Feed-Forward Layers For Modular Transfer Learning (1/3)

Adapters At A Glance



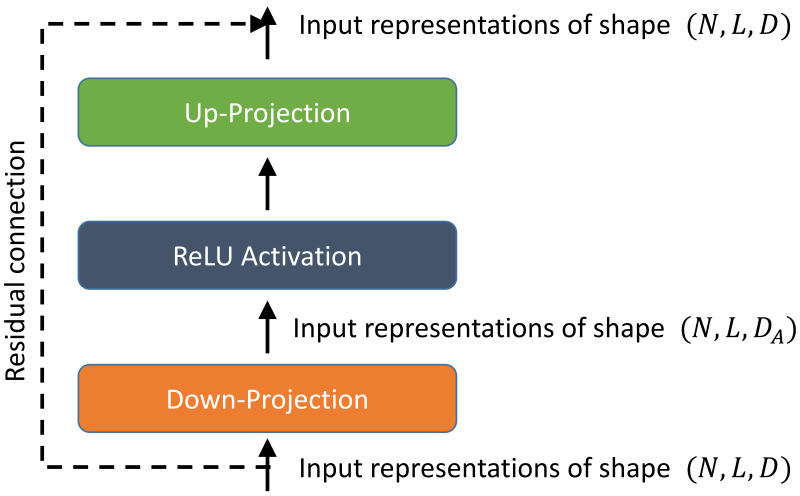
Comments

- **Desc:** Adapters are small feed-forward subnetworks (cf. next slide) typically added after pre-training at the end of each transformer block
- **Idea**: modularly isolate whatever information is key for `transfer’ (broad definition, transfer might be language, downstream task, etc., cf. **Adapt parameters**)
- **Performance:** expect slightly less than original FT (fine-tuning) performance

Adapters: Adapter Modules (2/3)

Lang En Adapt

Adapters At A Glance

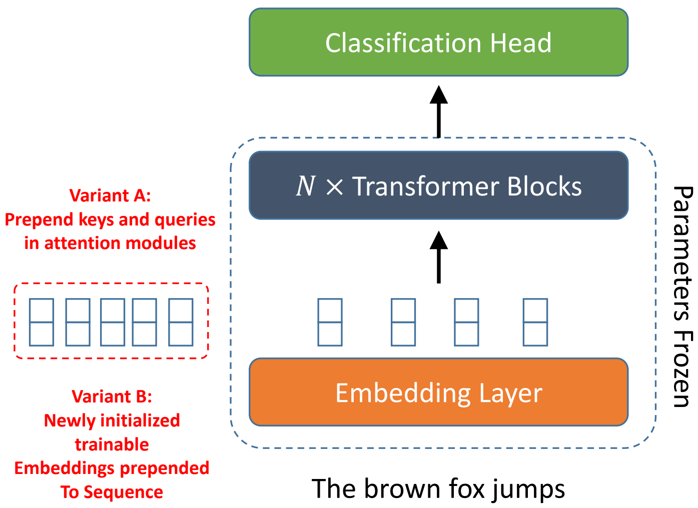


Comments

- **Down-Projection:** a linear layer that shares parameters across all token representations to down cast representations from dimensionality D (e.g. 768) to D_A (e.g. 16)
- **ReLU Activation:** required as we otherwise only learn lower rank (16 rank) approximations of original (eg 768 dimensional) representations – we need to learn to extract features meaningful for task!
- **Up:** a linear layer that shares parameters across all token representations to up cast representations from dimensionality D_A (e.g. 16) to D (e.g. 768)
- **Mind the residual connection!**
- **Parameter-efficiency:** $\mathbf{2 \times D \times D_A \ll 2 \times D^2}$ to reduce # parameters to $\frac{D_A}{D}$ of original parameters
- **Inference:** depth of transformer increases by number of adapters inserted; in practice, works just as well to omit “some” adapters. Expect 10-25% slower inference

Prefix-Tuning

Adapters At A Glance



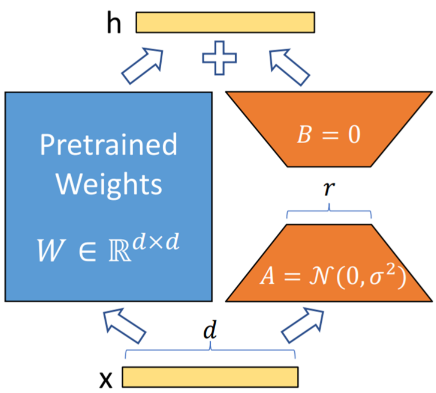
Comments

- **Desc:** prefix tuning prepends the sequence with additional embeddings that are learned to gear the token representations towards the task (cf. attention)
- **Idea**: the prefix embeddings are learned in a way such that, within attention modules, the token embeddings attend to prefix "tokens" to meaningfully update themselves for the task
- **Performance:** expect slightly less than original FT (fine-tuning) performance, some papers claim it might work better but sensitive to hyperparameters (prolonged training since initialized from scratch). This means high initial learning rate required, small final learning rate; not easy to bridge correctly between the two
- **Inference:** slowdown due to artificially increasing sequence length by number of prefixes; costly in attention (cf. quadratic complexity)

LoRA

LoRA At A Glance

Factorization commonly applied to keys, weights, and/or queries of attention matrices



$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Comments

- **Desc:** factorizes the parameter update of W (i.e. ΔW) into a low-rank series of down and up-projections
- **Idea**: parameter-efficient fine-tuning `works' because what tasks from downstream tasks is inherently `low-rank'; consequently, we can bend models to similar solutions as FT with lower rank
- **Performance: less** than original FT (fine-tuning) performance; problem is task-dependent on what parameters should be updated etc.
- **Inference:** no slow down because ΔW be merged into W to avoid overhead; training though slower comparable to adapters