Reducing the Computational and Memory Costs of Fine-tuning and Inference

Nicolas.Hernandez@univ-nantes.fr

M2 ATAL - 23 janvier 2025

Nantes Université

Laboratoire des Sciences du Numérique de Nantes

TALN

Overview

Quantization

Sparse attention methods

Knowledge distillation

Parameter Efficient Fine-Tuning (soft prompt and adapters)

Mixture of Experts

Quantization

Quantization

Basic idea of quantization:

> Going from high-precision representation for weights and activations to a lower precision data type

Requires less memory storage, consumes less energy (in theory), and operations like matrix multiplication can be performed much faster with integer arithmetic

Allows to run models on embedded devices, which sometimes only support integer data types

Quantization - Questions to ask

Important to consider the accumulation data type in order not to make the quantization process useless i.e. the type of the result of accumulating (adding, multiplying, etc) values of the data type in question

> Given two int8 values A = 127, B = 127,
> C as the sum of A and B = is much bigger than the biggest representable value in int8, which is 12

If further training is need for downstream tasks because training can be unstable due to the lower precision of the weights and activations

- Does my operation have an implementation of the targeted data type?
- Does my hardware support it?
- Is my operation sensitive to lower precision?
- What's the best way to project the range of the source data type values to the target data type space?
- How to calibrate i.e. determine the range of the source data type to project?

There are several ways to quantize a model

The traditional common quantization cases were:

- float32 (32-bit floating point)-> float16
- and float32 -> int8

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (Jacob et al. @Google, 2017)

There are several ways to quantize a model:

- optimizing which model weights are quantized with the AWQ algorithm

  AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration
  (Lin et al. @MIT, NVIDIA, IBM, UMass Amherst, MLSys 2024)

- independently quantizing each row of a weight matrix with the GPTQ algorithm

  GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers
  (Frantar et al. @IST, ETH, ICLR Mar 2023)

- quantizing to 8-bit and 4-bit precision with the bitsandbytes library
- quantizing to as low as 2-bit precision with the AQLM algorithm

  Extreme Compression of Large Language Models via Additive Quantization
  (Egiazarian et al. @HSE Univ, Yandex, Skoltech, IST, NeuralMagic, Sep 2024)

- optimize training and inference on targeted hardware (NVIDIA, AMD, Inter, AWS, Google TPU, ONNX...) with the Optimum Transformer extension

Constructing Transformers for longer sequences with Sparse Attention Methods

Original transformer cannot process long sequences

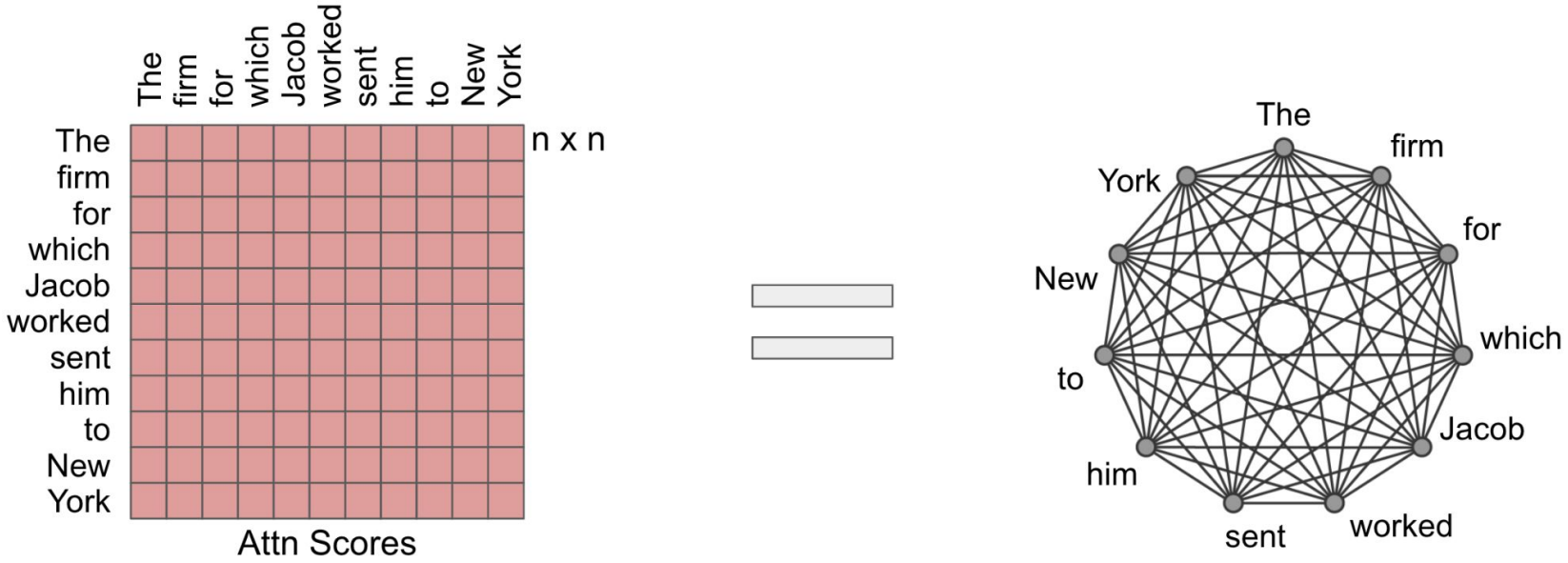
Problem:

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length

i.e. with O(n2) time and memory complexity where n is the input sequence length

Attention as a graph

Full attention can be viewed as a complete graph

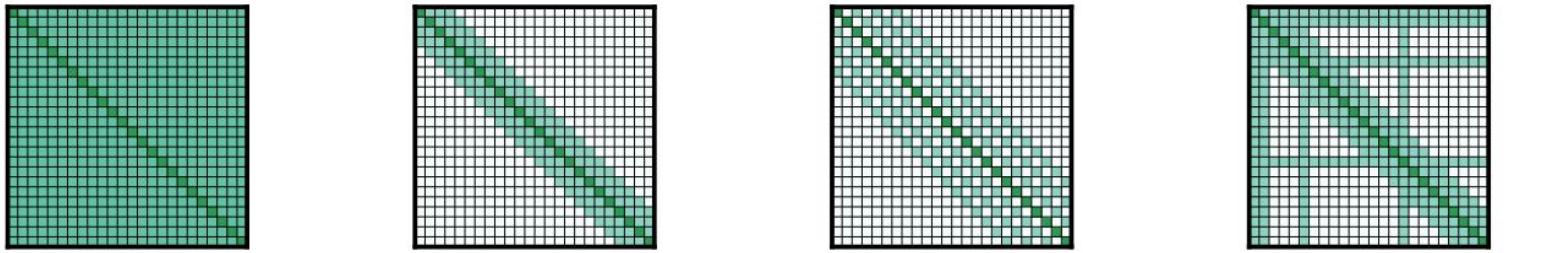


Source: Google blog

Longformer: The Long-Document Transformer (Beltagy et al. @allenai, Dec 2020)

Longformer, encoder with an attention mechanism that scales linearly with sequence length

Its attention pattern combines a local windowed attention with a task motivated global attention



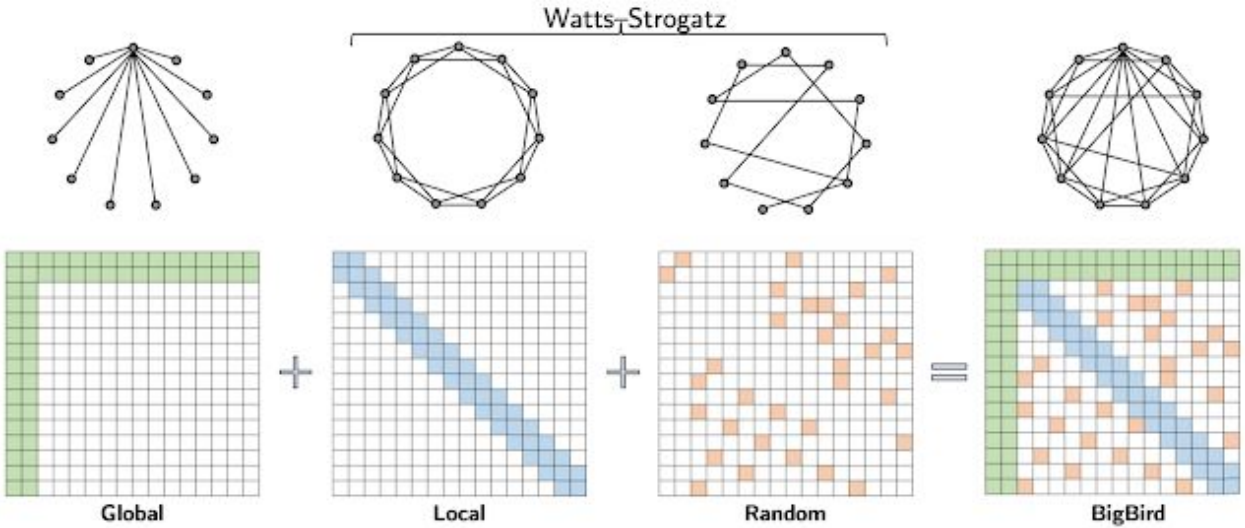
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Longformer-Encoder-Decoder (LED), variant for supporting long document generative tasks (effective on arXiv summarization dataset)

Big Bird: Transformers for Longer Sequences (Zaheer et al. @Google, NeurIPS Jan 2021)

BigBird sparse attention can be seen as adding few global tokens on Watts-Strogatz graph (i.e. graphs with small-word properties including short average path lengths and high clustering)



Model	MLM	SQuAD	MNLI
BERT-base	64.2	88.5	83.4
Random (R)	60.1	83.0	80.2
Window (W)	58.3	76.4	73.1
R + W	62.7	85.1	80.5
Global + R + W	64.4	87.2	82.9

Source: Google blog and slides

Long Range Arena : A Benchmark for Efficient Transformers (Tay et al. @Google, ICLR2021)

Tasks consisting of sequences ranging from 1K to 16K tokens, encompassing a wide range of data types and modalities such as text, natural, synthetic images, and mathematical expressions requiring similarity, structural, and visual-spatial reasoning

- ListOps (modeling hierarchically structured data)
  e.g. `INPUT: [MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]] OUTPUT: 5`
- Byte-level text classification (sentiment analysis)
- Byte-level document retrieval ability to compress long sequences into representations suitable for similarity-based matching
- Image classification on sequences of pixels
- Pathfinder

Long Range Arena : A Benchmark for Efficient Transformers (Tay et al. @Google, ICLR2021)

Current leaderboard results (as of 8th November 2020)

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Chance	10.00	50.00	50.00	10.00	50.00	50.00	44.00
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	54.39
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	**59.59**	**44.24**	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	76.34	FAIL	51.36
Reformer	**37.27**	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	36.99	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	59.29	40.83	74.87	FAIL	**55.01**
Linear Trans.	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	65.40	53.82	42.77	**77.05**	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

Table 1: Experimental results on Long-Range Arena benchmark. Best model is in boldface and second best is underlined. All models do not learn anything on Path-X task, contrary to the Pathfinder task and this is denoted by FAIL. This shows that increasing the sequence length can cause seriously difficulties for model training. We leave Path-X on this benchmark for future challengers but do not include it on the Average score as it has no impact on relative performance. **Important: These**

Long Range Arena : A Benchmark for Efficient Transformers (Tay et al. @Google, ICLR2021)

March 2021 reran all models for the ListOps

Train models longer up to 10K steps

Performer and Linformer, comparable results

Performer the 2nd best model

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Chance	10.00	50.00	50.00	10.00	50.00	50.00	44.00
Transformer	36.38	64.27	57.46	42.44	71.40	FAIL	54.39
Local Attention	15.95	52.98	53.39	41.46	66.63	FAIL	46.08
Sparse Trans.	35.78	63.58	**59.59**	**44.24**	71.71	FAIL	54.98
Longformer	36.03	62.85	56.89	42.22	69.71	FAIL	53.54
Linformer	35.49	53.94	52.27	38.56	76.34	FAIL	51.32
Reformer	36.30	56.10	53.40	38.07	68.50	FAIL	50.47
Sinkhorn Trans.	34.20	61.20	53.83	41.23	67.45	FAIL	52.78
Synthesizer	36.50	61.68	54.67	41.61	69.45	FAIL	52.78
BigBird	**37.08**	64.02	59.29	40.83	74.87	FAIL	**55.22**
Linear Trans.	17.15	**65.90**	53.09	42.34	75.30	FAIL	50.76
Performer	36.00	65.40	53.82	42.77	**77.05**	FAIL	55.01
Task Avg (Std)	32 (7.9)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

Table 6: Experimental results on Long-Range Arena benchmark. Best model is in boldface and second best is underlined. All models do not learn anything on Path-X task, contrary to the Pathfinder task and this is denoted by FAIL. This shows that increasing the sequence length can cause seriously difficulties for model training. We leave Path-X on this benchmark for future challengers but do not include it on the Average score as it has no impact on relative performance.

Long Range Arena : A Benchmark for Efficient Transformers (Tay et al. @Google, ICLR2021)

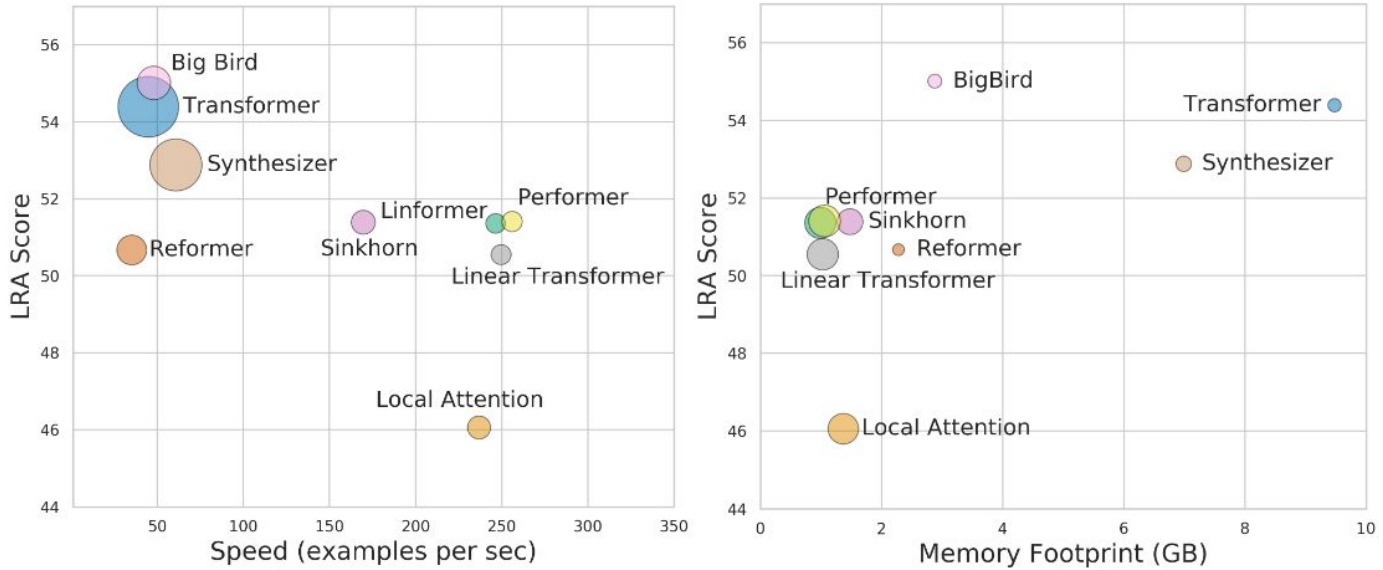


Figure 3: Trade-off between performance (y axis) and resources (x axis). On the left, circle size corresponds to memory footprint. On the right, it corresponds to examples per second.

ERNIE-SPARSE: Learning Hierarchical Efficient Transformer Through Regularized Self-Attention (Liu et al. @baidu, Mar 2022)

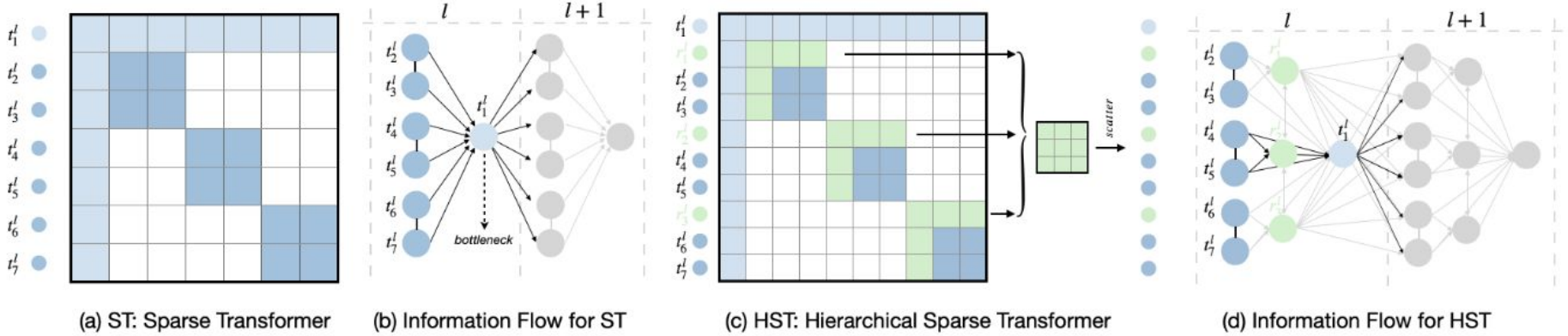


Figure 1: **The comparison between Sparse Transformer (ST) and Hierarchical Sparse Transformer (HST).** **(a)** Sparse Transformer mainly consists of global attention and local attention. **(b)** The bottleneck that existed in ST: all the sequence information is compressed into a fixed size vector. **(c)** In HST, representative tokens are inserted into local attention for hierarchical attention. **(d)** Information flow demonstration for HST: interaction between representative nodes can increase the path of global information interaction.

Knowledge Distillation

credit: flaticon

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter (Sanh et al. @huggingface, NeurIPS 2019)

Method (and model name) :

- Given a large general-purpose language representation model called teacher (BERT),
- pre-train a student model with same general architecture but simpler (DistilBERT),
- with the following triple objectives:
  - find the same distribution probabilities as the teacher model
  - predict the masked tokens correctly (but no next-sentence objective)
  - a cosine similarity between the hidden states of the student and the teacher model

40% less parameters (# of layers reduced by 2 + less components e.g. segment-type embedding), trained on the same data,
obtains 95% of its performances (when fine-tuned on GLUE tasks), 60% faster in inference time

Parameter Efficient Fine-Tuning (PEFT)

Parameter-Efficient Fine-Tuning (PEFT)

Problem: Need to adapt LLM to specific tasks but

- full LLM fine-tuning and storage have prohibitive computational and memory costs
  - In addition to model weights (hundreds of gigabytes for the largest models), must allocate memory for optimizer states, gradients, forward activations, and temporary memory throughout the training process
- cannot be done on consumer hardware
- and as many full fine-tuning to perform as tasks to handle

Main idea: only fine-tune (update) a small number of (extra) model parameters

- 1) freeze some layer or components, fine tuning the remaining, or 2) freeze all the pre-trained and add small number of parameters or layers and fine tune only the new components

The Hugging Face peft library and the adapters library from Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning (Poth et al, EMNLP2023) implement several methods

PEFT approach advantages

- makes fine-tuning more efficient by drastically reducing the number of trainable parameters (reduce time and memory costs)
- multiple lightweight and portable PEFT models for various downstream tasks or simple hardware (single GPU or embedding devices)
- often allow combination of multiple PEFT methods
- performance comparable to fully fine-tuned models
- recycle a single pre-trained model for all downstream tasks
- retain the efficient serving benefits of frozen model i.e. avoid the catastrophic forgetting issue
- allow to train a quantized model with a PEFT adapter

PEFT method families

- soft prompting: add learnable parameters to the input embeddings, aka "virtual tokens" (e.g. prefix tuning, prompt tuning, p-tuning)
- adapter addition: 1) add extra layers between some neural blocks or 2) express weight updates as a low-rank decomposition of the delta weight matrix or 3) multiply model's activations by learned vectors of specific neural blocks (e.g. LoRA)

Soft prompting

Prefix-Tuning: Optimizing Continuous Prompts for Generation (Li and Liang @Stanford, Jan 2021)

Method:

- keep positions for a prefix prompt and learn the matrix Pθ assumed output of activation functions for these positions (autoregressive has one prefix and encoder-decoder two)
- $$h_i = \begin{cases} P_\theta[i,:], & \text{if } i \in \mathsf{P}_{\mathsf{idx}}, \\ \mathrm{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

  - Pθ (parametrized by θ), trainable matrix of dimension |Pidx| × dim(hi)
  - z, concatenation of x and y till the i position,
  - LM, language model with frozen parameters ϕ,
  - $h_{<i}$ values of activation functions of previous i
  - added to every layer of the model

Applied to GPT-2 for table-to-text generation and to BART for summarization

Learning only 0.1% of the parameters, obtains comparable performance, and extrapolates better to examples with topics unseen during training

Prefix-Tuning: Optimizing Continuous Prompts for Generation (Li and Liang @Stanford, Jan 2021)

(3)

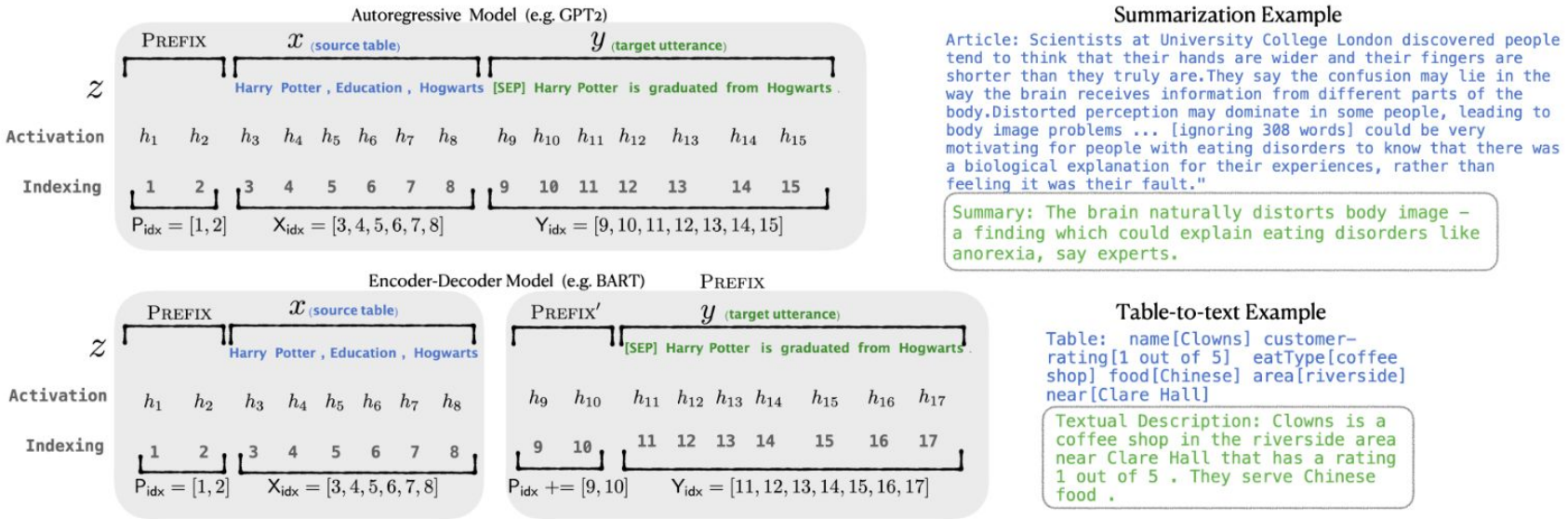


Figure 2: An annotated example of prefix-tuning using an autoregressive LM (top) and an encoder-decoder model (bottom). The prefix activations $\forall i \in \mathsf{P}_{\mathsf{idx}}, h_i$ are drawn from a trainable matrix P_θ. The remaining activations are computed by the Transformer.

The Power of Scale for Parameter-Efficient Prompt Tuning (Lester et al. @Google Research, Apr 2021)

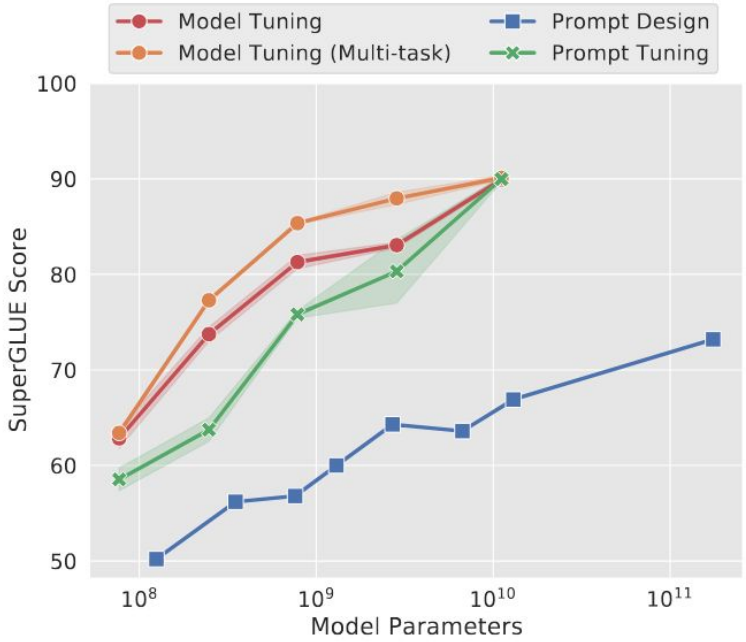
Method :

- Add k tunable tokens per downstream task to be prepended to the input text, learn the embeddings of these prompt tokens
- Various initialization strategies explored (random, with embeddings that enumerate the output classes for classification tasks...)

Applied to T5 XXL model, competitive to full fine-tuning (called here model tuning) as the prompt tuned model exceed billions of parameters ; reduction over 5 orders of magnitude with a prompt length of 5 tokens

Outperforms GPT-3's "few-shot" learning by a large margin

The Power of Scale for Parameter-Efficient Prompt Tuning ([Lester et al. @Google Research, Apr 2021](#))



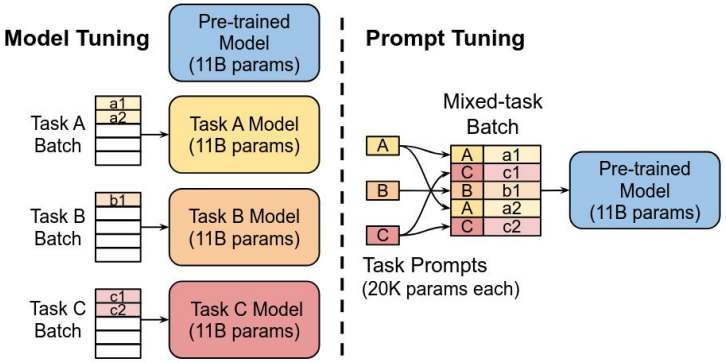


Figure 2: **Model tuning** requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches. **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pre-trained model. With a T5 "XXL" model, each copy of the tuned model requires 11 billion parameters. By contrast, our tuned prompts would only require 20,480

Adapters

LoRA: Low-Rank Adaptation of Large Language Models (Hu et al. @Microsoft, Oct 2021)

Method :

- Assume that the weight updates ΔW (i.e. the task-specific parameters) can be represented with two smaller matrices (called update matrices) through low-rank decomposition

  $h = W_0 x + \Delta W x = W_0 x + BAx$
- Set rank r, freeze pre-trained weights, fine-tune B and A

With

- $W_0 \in R^{d \times k}$, the frozen pre-trained weight matrix
- $B \in R^{d \times r}$, $A \in R^{r \times k}$,
- and the rank $r = \min(d, k)$

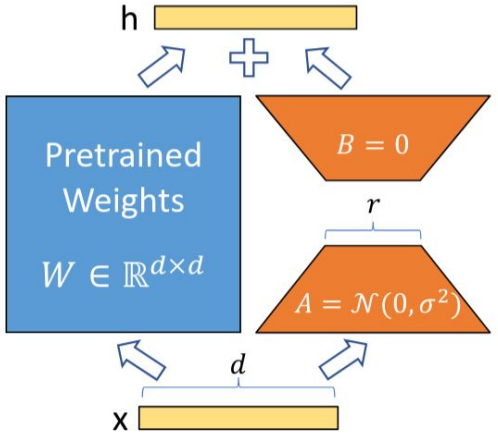


Figure 1: Our reparametrization. We only train A and B.

LoRA: Low-Rank Adaptation of Large Language Models (Hu et al. @Microsoft, Oct 2021)

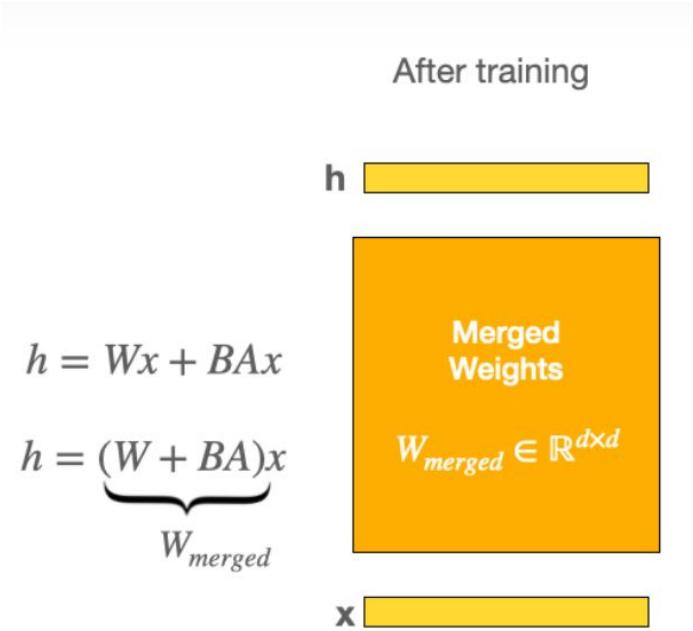
Can be applied to any weight matrices in a neural network but in practice only applied to the attention blocks weights

Initialized with `B=0` and `A` with a normal law/distribution (a bell curve) $\mathcal{N}(0, \sigma^2)$ i.e. with a mean μ (mu) centered on 0 and σ^2 (squared sigma) as variance

Various size of r experimented from 1 to 64

Original and adapter weights combined by addition ; adapter and pre-trained weights can be merged to eliminate inference latency

Resulting number of trainable parameters depends on the size of the update matrices, which is determined mainly by the rank r and the shape of the original weight matrix



IA3

TODO

Mixture of experts

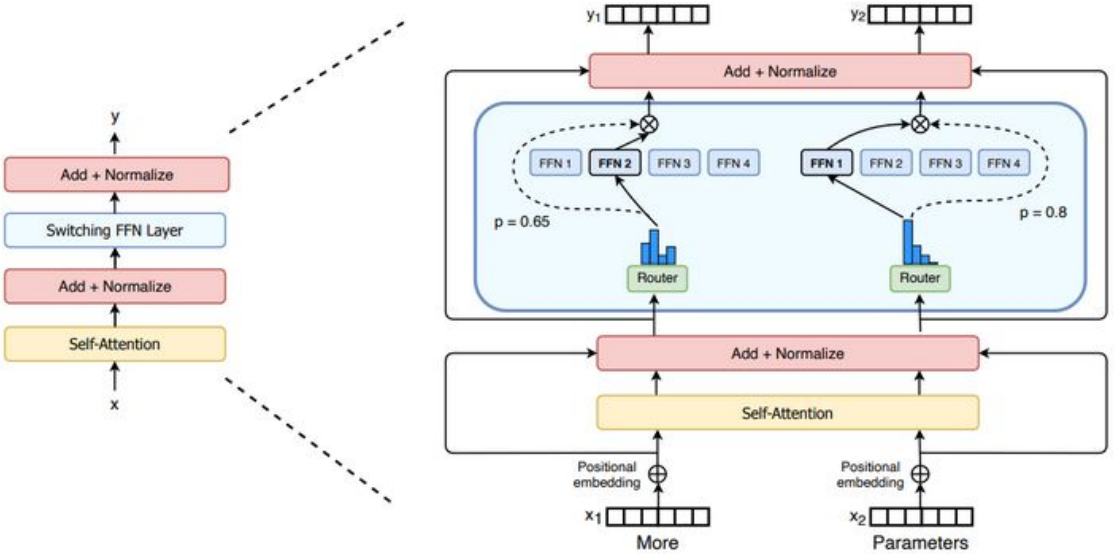
Mixture of Experts (MoE)

Two main elements:

- Sparse MoE layers: replaces a dense feed-forward network (FFN) layer with a certain number of “experts” (e.g. 8) which are usually are FFNs (can be more complex like MoEs too)
- A gate network or router: determines which tokens are sent to which expert.
  A token can be sent to more than one expert
  Router parameters learnt during pre-training stage

Mixture of Experts (MoE)

Here the token “More” sent to the second expert, and token "Parameters” sent to the first network



Source: MoE layer from the [Switch Transformers paper](https://arxiv.org/abs/2101.03961)

Mixture of Experts (MoE)

Training:

- significantly more compute-efficient pretraining,
- but historically have struggled to generalize during fine-tuning, leading to overfitting

Inference:

- Although have many parameters, only some of them are used during inference (some are shared and some are the ones from the experts involved)
- Much faster inference compared to a dense model with the same number of parameters
- However, all parameters need to be loaded in RAM, so memory requirements are high

See HF Mixtral-8x7B doc (and Mistral.ai blog) the MoE version of Mistral-7b (HF doc and Mistral.ai blog)

Conclusion

TODO

TODO