S-LORA – Serving Thousands of Concurrent LORA Adapters.

(PRETRAIN THEN FINE TUNE) Commonly adopted in LLM

LORA

Low Rank Adaption

PARAMETER EFFICIENT FINE TUNING METHOD

Understand What LORA is all about

GPT 3 175B → Out to fine tune it own all 175 B from start

Pre trained with huge Data

Which is less feasible and prohibitively expensive.

To Recover that

LORA (Low Rank Adaption.)

① freezes the pre trained Model weights

② injects the trainable rank decomposition matrices into each layer of transformer Architecture

Low Rank adaptive Matrices greatly reduces the No. of Parameters to Down line

adapter weights GPT 3 175B

LORA Can reduce the trainable parameter 10,000 times than the Adam GPU Memory Requirement By 3 times

As an inspiration they took =)

[learned over parameterized models in fact reside on low" intrinsic dimension

Now Changes in Weights during Model Adaption

⇓

Low Intrinsic Rank

LoRA ⟹ [train only some dense layers in a NN] indirectly optimizes the Rank decomposition matrices of the dense layer. while keeping the pre trained weights frozen

pre trained Auto Regressive Language Model.

$P_\Phi(y|x)$ → Parameter

Can be a generic multi task learner such as GPT.

Each Downstream task is represented by the training dataset

$Z = \{(x_i, y_i), \ldots\}_{i=1,\ldots N}$

x_i – input y_i ~ target (output)

During full fine tuning:

Model

$P_\Phi(y|x)$ ⟹ Pre trained with Φ

and updated to $\Phi + \Delta\Phi$ by repeatedly following the gradient to maximize the conditional Language modelling objective.

y_t

$$\max_{\Phi} \sum_{(x,y)\in Z} \sum_{t=1} \log\left(P_{\Phi}(y_t \mid x, y_{<t})\right)$$

Drawbacks during full fine tuning:

- May learned different set of parameter $\Delta\Phi$ whose dimension $|\Delta\Phi| = |\Phi|$
- 175B — the tuned size
- 125B — Pre-trained size

LoRA $\Delta\Phi = \Delta\Phi(\Theta)$

Small sized set of Parameters Θ with $|\Theta| \ll |\Phi_0|$

Dimension $|Y|$

$$\max_{\Theta} \sum_{(x,y)\in Z} \sum_{t=1} \log\left(P_{\Phi_0 + \Delta\Phi(\Theta)}(y_t \mid x, y_{<t})\right)$$

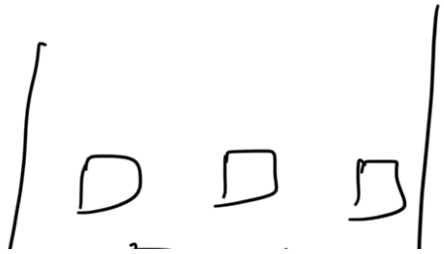
LoRA ⟹ Base one Model

Substantial Collection of **LoRA Adapters**

Advantage ⟸

S-LoRA ⟹ Serving Concurrent of Thousand Adapters.

SLoRA



fetch the adapter that are

all the adapters in the main memory

currently running query to GPU Memory

To efficiently use GPU & Reduce fragmentation ⇒ S-LoRA [Unified Paging]

① Unified Memory Pool ← ①

to manage dynamic adapter weights

Stores both loading adapter weights paged KV Cache. in a

② Tensor Parallelism

③ Highly optimized CUDA kernels

Heterogeneous Batch LoRA Computation.

Compared with [Hugging face PEFT, vLLM] & S-LoRA improves throughput by upto 4 times and increases the number of served adapters.

LoRA Low rank update

① Low Rank Parametrized Update Matrices:

NN many Dense layers ⇒ Matrix Multiplication

Weight Matrices in these layers ⇒ typically have full Rank.

Low intrinsic Rank dimensionality adaptation

Pretrained Weight Matrix $W_0 \in \mathbb{R}^{d \times k}$

$W_0 + \Delta W = W_0 + BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$ and the rank $r \leq \min(d,$

latter

Constraint

During training: freezes W_0 => will not receive any gradient updates.

A and B Contains trainable Parameters.

Note both W_0 and $\Delta W = BA$ are multiplied with the same input and their respective vector are summed coordinate wise.

$h = W_0 x$ The Modified forward pass into the transformer

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

Random Gaussian initialization for A
zero for B

at the

so $\Delta W = BA = 0$ beginning of the training.

(2) Then scale $\Delta W x$ by $\frac{\alpha}{r}$ where α is the constant in r

(2) Applying LoRA to transformer.

We know that in transformer there are 4 Weight Matrices.

W_q W_k W_v W_o and two in MLP Module

They treat $[W_q \circ [W_k \; W_v \; W_o]]$ as a single matrix of dimension. $[d_{model} \times d_{model}]$ the output is usually even though sliced into attention heads.