AI Applications Lecture 18
Parameter Efficient Fine Tuning and LoRA

SUZUKI, Atsushi
Jing WANG
2025-11-18

Contents

1 Introduction

The machine learning framework used in much of generative AI is divided into two steps: the ”learning” step, which appropriately determines the parameters of a parametric function, and the ”inference” step, which actually executes the desired task using the function determined by the parameters set in the learning step. This lecture, like the previous one, focuses on ”learning”.

The neural networks currently used in practice in the field of generative AI have an enormous number of parameters. Since machine learning is a framework for determining the parameters of a parametric function from training data, an amount of training data corresponding to the number of parameters to be determined is required. For example, if one attempts to minimize an objective function using a gradient method, the objective function itself must contain enough information to sufficiently judge the quality of the parameters for the task one wants to solve, and it is easy to understand that this requires an amount of training data corresponding to the number of parameters.

Collecting a large amount of training data is accompanied by difficulties, and even if it is collected, learning on it requires a large amount of computation. Therefore, if possible (i.e., if there are no major performance problems), one would like to reduce the amount of training data used. For learning to succeed even with less training data, the number of parameters changed during learning needs to be small (of course, this is a necessary condition for successful learning, not a sufficient condition).

For the reasons mentioned above, **one would like to reduce the number of parameters changed during learning, if possible**. Of course, in a situation where one is building a model by training from scratch, if the task itself is complex and the desired input-output relationship is complicated, a high-dimensional checkpoint (a large number of parameters) and a large-scale architecture are necessary to represent that complex input-output relationship, and data for training it is also necessary. Thus, the policy of reducing the number of parameters to be learned is inherently unreasonable.

On the other hand, if one is in a situation where one **already has** a model f_θ that gives a good input-output relationship, and it is expected that a better input-output relationship can be obtained by modifying it slightly (in some sense), then a scheme of **learning the minimum number of parameters just enough to represent the part corresponding to that slight modification** might be practically feasible. The idea of starting from an existing model and modifying it slightly to obtain a better model is called **fine-tuning**, and the idea of realizing this by learning a small number of parameters is called **PEFT (parameter-efficient fine-tuning)**. In this lecture, we will learn about **LoRA (low-rank adaptation)** as a typical example of PEFT [1, 2].

1.1 Learning Outcomes

After completing this lecture, students should be able to do the following:

- Control the behavior of generative AI using LoRA.
- Explain the advantages of LoRA.
- Distinguish and explain what LoRA does mathematically and what it does in implementation.

2 Preparation: Mathematical Notations Revisited

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a \coloneqq b$ indicates that a is defined by b.
- **Set (Set):**
  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that the element x belongs to the set $\mathcal{A}$.
  - $\{\}$: Empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (extensional notation).
  - $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in set $\mathcal{A}$ for which the proposition $P(x)$ is true (intensional notation).
  - $\mathbb{R}$: The set of all real numbers.
  - $\mathbb{R}_{>0}$: The set of all positive real numbers.
  - $\mathbb{R}_{\geq 0}$: The set of all non-negative real numbers.
  - $\mathbb{Z}$: The set of all integers.
  - $\mathbb{Z}_{>0}$: The set of all positive integers.
  - $\mathbb{Z}_{\geq 0}$: The set of all non-negative integers.
  - $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \ldots, k\}$: For a positive integer k, the set of integers from 1 to k.
- **Function (Function):**
  - $f : \mathcal{X} \to \mathcal{Y}$: Indicates that the function f is a map that takes elements from set $\mathcal{X}$ as input and outputs elements from set $\mathcal{Y}$.
  - $y = f(x)$: Indicates that the output is $y \in \mathcal{Y}$ when $x \in \mathcal{X}$ is input to the function f.
- **Vector (Vector):**

- In this course, a vector refers to a column of numbers.
- Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$.
- $\boldsymbol{v} \in \mathbb{R}^n$: Indicates that the vector $\boldsymbol{v}$ is an n-dimensional real vector.
- The i-th element of vector $\boldsymbol{v}$ is denoted as v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \tag{1}$$

- **Matrix (Matrix):**
  - Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A}$.
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: Indicates that the matrix $\boldsymbol{A}$ is an $m \times n$ real matrix.
  - The element in the i-th row and j-th column of matrix $\boldsymbol{A}$ is denoted as $a_{i,j}$.

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}. \tag{2}$$

  - The transpose of matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^\top$. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$, then $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$, and

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix} \tag{3}$$

    .
  - A vector is also a matrix with 1 column, and its transpose can also be defined.

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{1,n} \tag{4}$$

    .

- **Tensor (Tensor):**
  - In this lecture, the term tensor simply refers to a multi-dimensional array. A vector can be regarded as a 1st-order tensor, and a matrix as a 2nd-order tensor. Tensors of 3rd-order or higher are denoted by underlined bold italic uppercase letters, like $\underline{\boldsymbol{A}}$.

– Students who have already learned about abstract tensors in mathematics or physics may feel resistant to calling a mere multi-dimensional array a tensor. If one considers that the basis is always fixed to the standard basis and the mathematical tensor is identified with its component representation (which becomes a multi-dimensional array), then there is (arguably) consistency in the terminology.

3 The Idea of PEFT: Representing Model Differences with Low-Dimensional Parameters

3.1 Review of PEFT Motivation

Let's review the motivation for PEFT. We assume a situation where we already have a model f_{θ} that provides a good input-output relationship, and we expect to obtain a better input-output relationship by modifying it slightly. In this situation, the idea of PEFT is to try to represent the part corresponding to that slight modification with a small number of parameters (or a low-dimensional parameter vector).

Remark 3.1. Here, "slight modification" is not something to be defined strictly mathematically. Although there are many mathematical criteria for measuring the closeness of functions, it does not necessarily mean "slight" in that sense here. "Slight" here, in the end, can only be defined as a modification that can be represented by a small number of parameters, which becomes a tautology. To be more precise, it is more accurate to say that methods have been empirically found that can achieve good performance improvements with specific modifications represented by a small number of parameters, and these are called PEFT.

3.2 General Definition of PEFT Adapters

We will strictly define PEFT as a function. Here, we let the parameter space of the original model f_{θ} be

$$\boldsymbol{\theta} \in \mathbb{R}^{d_{\text{Original}}} \tag{5}$$

and the input and output spaces be $\mathcal{X}$ and $\mathcal{Y}$, respectively.

Definition 3.1 (PEFT Adapter)**.** Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the output space. Fix $d_{\text{Original}} \in \mathbb{Z}_{>0}$, and suppose we are given a family of functions with parameters on the parameter vector space $\mathbb{R}^{d_{\text{Original}}}$

$$f_{(\cdot)} : \mathbb{R}^{d_{\text{Original}}} \to \mathcal{Y}^{\mathcal{X}} \tag{6}$$

. Here, $f_{(\cdot)}$ is a "map from parameters to models", and given $\boldsymbol{\theta} \in \mathbb{R}^{d_{\text{Original}}}$, a specific function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ is determined.
Fix $d_{\text{PEFT}} \in \mathbb{Z}_{>0}$, and let the **PEFT parameter** be $\boldsymbol{\psi} \in \mathbb{R}^{d_{\text{PEFT}}}$. At this time, a **PEFT adapter**

is a map

$$\mathrm{PEFTAdapter}_{f_{(\cdot)},\boldsymbol{\psi}} : \mathbb{R}^{d_{\mathrm{Original}}} \to \mathcal{Y}^{\mathcal{X}} \tag{7}$$

. The original model $\boldsymbol{\theta} \in \mathbb{R}^{d_{\mathrm{Original}}}$ is transformed into

$$\mathrm{PEFTAdapter}_{f_{(\cdot)},\boldsymbol{\psi}}(\boldsymbol{\theta}) \tag{8}$$

, depending on the small number of parameters $\boldsymbol{\psi}$. The specific way this transformation is determined depends on the PEFT method.

Remark 3.2. Definition 3.1 defines PEFT very abstractly, but the essence is as follows. We fix a base model $f_{\boldsymbol{\theta}}$ and construct a new function $\mathrm{PEFTAdapter}_{f_{(\cdot)},\boldsymbol{\psi}}(\boldsymbol{\theta})$ through the PEFTAdapter. At this time, $\mathrm{PEFTAdapter}_{f_{(\cdot)},\boldsymbol{\psi}}(\boldsymbol{\theta})$ is controlled by the low-dimensional parameter $\boldsymbol{\psi}$. That is, it is a framework that allows access to various models in the vicinity of $f_{\boldsymbol{\theta}}$ by changing $\boldsymbol{\psi}$.

Under Definition 3.1, **PEFT is a framework for determining the PEFT parameter $\boldsymbol{\psi}$ in the PEFT adapter by learning, so that it becomes a "good" function for the desired task**. More specifically, the learning in PEFT involves defining some objective function (e.g., a loss function) as a function of $\boldsymbol{\psi}$ and minimizing it.

3.3 A Simple Example of PEFT

Since Definition 3.1 is abstract, let's consider the simplest specific example. Consider the case where the original parameter vector $\boldsymbol{\theta}$ is explicitly divided into a part that is not updated during learning and a part that is updated.

Example 3.1 (Simple PEFT by Modifying Only Some Parameters)**.** For $d_{\mathrm{fixed}}, d_{\mathrm{tuned}} \in \mathbb{Z}_{>0}$, let

$$d_{\mathrm{Original}} \coloneqq d_{\mathrm{fixed}} + d_{\mathrm{tuned}} \tag{9}$$

and represent the parameter vector by block partitioning as

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_{\mathrm{fixed}} \\ \boldsymbol{\theta}_{\mathrm{tuned}} \end{bmatrix} \in \mathbb{R}^{d_{\mathrm{Original}}} \tag{10}$$

. Here, suppose $\boldsymbol{\theta}_{\mathrm{fixed}} \in \mathbb{R}^{d_{\mathrm{fixed}}}$ are the parameters not changed during learning (fixed parameters), and $\boldsymbol{\theta}_{\mathrm{tuned}} \in \mathbb{R}^{d_{\mathrm{tuned}}}$ are the parameters changed during learning.
At this time, we define the **simple PEFT adapter** $\mathrm{SimplePEFTAdapter}_{f_{(\cdot)},\Delta\boldsymbol{\theta}_{\mathrm{tuned}}}$ as follows:

$$\mathrm{SimplePEFTAdapter}_{f_{(\cdot)},\Delta\boldsymbol{\theta}_{\mathrm{tuned}}}\left(\begin{bmatrix} \boldsymbol{\theta}_{\mathrm{fixed}} \\ \boldsymbol{\theta}_{\mathrm{tuned}} \end{bmatrix}\right) \coloneqq f_{\begin{bmatrix} \boldsymbol{\theta}_{\mathrm{fixed}} \\ \boldsymbol{\theta}_{\mathrm{tuned}} + \Delta\boldsymbol{\theta}_{\mathrm{tuned}} \end{bmatrix}}. \tag{11}$$

Here, $\Delta\boldsymbol{\theta}_{\mathrm{tuned}} \in \mathbb{R}^{d_{\mathrm{tuned}}}$ is the difference parameter determined by learning, and it corre-

sponds to the PEFT parameter in this example. That is, matching the notation of Definition 3.1, we have

$$\boldsymbol{\psi} = \Delta\boldsymbol{\theta}_{\text{tuned}}, \quad d_{\text{PEFT}} = d_{\text{tuned}} \tag{12}$$

.

Remark 3.3. Let's make some observations from Example 3.1.

- First, the number of parameters to be determined is $d_{\text{PEFT}} = d_{\text{tuned}}$, which can be set freely by the designer. In particular, it can be designed such that $d_{\text{tuned}} \ll d_{\text{Original}}$. This is a great advantage in terms of learning stability when the available training data is scarce.

- Next, consider the perspective of memory for storage. If one only wants to obtain the result of PEFT, one may discard the original $\boldsymbol{\theta}$ and save only

$$\begin{bmatrix} \boldsymbol{\theta}_{\text{fixed}} \\ \boldsymbol{\theta}_{\text{tuned}} + \Delta\boldsymbol{\theta}_{\text{tuned}} \end{bmatrix} \tag{13}$$

  . The number of parameters to save in this case is d_{Original}, which is the same as saving the original model.

- On the other hand, suppose that both the original $f_{\boldsymbol{\theta}}$ and the PEFT-modified $f_{\begin{bmatrix} \boldsymbol{\theta}_{\text{fixed}} \\ \boldsymbol{\theta}_{\text{tuned}} + \Delta\boldsymbol{\theta}_{\text{tuned}} \end{bmatrix}}$ are valuable for applications. In this case, naively thinking, one would save two sets of parameters, $\boldsymbol{\theta}$ and (13), resulting in storing a total of $2d_{\text{Original}}$ floating-point values.

- However, in reality, the PEFT-modified (13) can be recovered from the original $\boldsymbol{\theta}$ and $\Delta\boldsymbol{\theta}_{\text{tuned}}$ as

$$\begin{bmatrix} \boldsymbol{\theta}_{\text{fixed}} \\ \boldsymbol{\theta}_{\text{tuned}} + \Delta\boldsymbol{\theta}_{\text{tuned}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_{\text{fixed}} \\ \boldsymbol{\theta}_{\text{tuned}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \Delta\boldsymbol{\theta}_{\text{tuned}} \end{bmatrix} \tag{14}$$

  , so the number of floating-point values that need to be saved is

$$d_{\text{Original}} + d_{\text{PEFT}} = d_{\text{Original}} + d_{\text{tuned}} \tag{15}$$

  . Usually, $d_{\text{tuned}} \ll d_{\text{Original}}$, so PEFT can significantly reduce the number of floating-point values to be saved.

- In particular, if PEFT is performed separately for multiple purposes, for example, if K PEFTs are performed and each PEFT result is valuable for applications. Naive storage would require saving

$$(K+1)d_{\text{Original}} \tag{16}$$

floating-point values. On the other hand, when using PEFT, one only needs to save the original $\boldsymbol{\theta}$ and each task-specific $\Delta\boldsymbol{\theta}^{(k)}_{\text{tuned}}$ $(k = 1, \ldots, K)$, so the number of floating-point values to be saved is

$$d_{\text{Original}} + K d_{\text{PEFT}} = d_{\text{Original}} + K d_{\text{tuned}} \tag{17}$$

. When $d_{\text{tuned}} \ll d_{\text{Original}}$ and K is large, this difference is practically very significant.

Thus, PEFT is advantageous from both the perspective of "number of parameters to update during learning" and "number of parameters to save". In the next chapter, we will introduce LoRA, which achieves even higher parameter efficiency by imposing further structural constraints on parameters represented as matrices.

4 The Idea of Low Rank Adaptation (LoRA)

4.1 Low-Rank Constraint on the Difference Representation

In the method described above, if $d_{\text{PEFT}} \ll d_{\text{Original}}$, the parameter efficiency of PEFT is good, but a concern is whether it can provide practically good differences when d_{PEFT} is small. Of course, mathematically considering the worst case, it's impossible for a small number of parameters to substitute for many parameters. The issue here is not the mathematical worst case, but whether, when considering practical utility in natural or social sciences, a better difference can be represented with a small number of parameters.

In fact, for the neural networks of generative AI in practical use, it is empirically known that when a part of the parameters can be regarded as a matrix representing a linear transformation, it is better to limit the difference to be represented by a **low-rank matrix** using fewer parameters, rather than representing the difference directly. This can represent practically useful differences despite the small number of parameters [1, 2]. **LoRA (low-rank adaptation)** is based on this idea, representing the difference for matrix-form parameters as a product of a matrix with few rows (a matrix with a small number of rows) and a matrix with few columns (a matrix with a small number of columns).

As a basic matter of linear algebra, we will strictly define the rank of a matrix based on matrix factorization.

Definition 4.1 (Definition of Matrix Rank by Factorization)**.** Let $d_{\text{out}}, d_{\text{in}} \in \mathbb{Z}_{>0}$ and $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}}, d_{\text{in}}}$. Consider the set

$$S(\boldsymbol{W}) \coloneqq \left\{ r \in \mathbb{Z}_{\geq 0} \,\middle|\, \exists\, \boldsymbol{B} \in \mathbb{R}^{d_{\text{out}}, r},\ \boldsymbol{A} \in \mathbb{R}^{r, d_{\text{in}}} \text{ exist such that } \boldsymbol{W} = \boldsymbol{B}\boldsymbol{A} \right\} \tag{18}$$

. At this time, the **rank** of $\boldsymbol{W}$ is defined as

$$\operatorname{rank}(\boldsymbol{W}) \coloneqq \min S(\boldsymbol{W}) \tag{19}$$

. That is, $\mathrm{rank}(\boldsymbol{W})$ is the minimum value of the number of column vectors r required when representing $\boldsymbol{W}$ as

$$\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}, \quad \boldsymbol{B} \in \mathbb{R}^{d_{\mathrm{out}},r}, \ \boldsymbol{A} \in \mathbb{R}^{r,d_{\mathrm{in}}} \tag{20}$$

, and it represents the "minimum number of vectors" necessary to represent $\boldsymbol{W}$. In particular, if $\boldsymbol{W} = \boldsymbol{0}$, then $\mathrm{rank}(\boldsymbol{W}) = 0$.

Proposition 4.1 (Equivalent Characterizations of Matrix Rank)**.** For $\mathrm{rank}(\boldsymbol{W})$ defined in Definition 4.1, the following hold.

(1) $\mathrm{rank}(\boldsymbol{W}) = \dim \mathcal{C}(\boldsymbol{W})$. Here, $\mathcal{C}(\boldsymbol{W})$ is the subspace spanned by the column vectors of $\boldsymbol{W}$, called the **column space**.

(2) $\mathrm{rank}(\boldsymbol{W}) = \dim \mathcal{R}(\boldsymbol{W})$. Here, $\mathcal{R}(\boldsymbol{W})$ is the subspace spanned by the row vectors of $\boldsymbol{W}$, called the **row space**.

In particular, the dimensions of the column space and row space of $\boldsymbol{W}$ are always equal.

Proof. First, we show (1). Let $\mathrm{rank}(\boldsymbol{W}) = r$. By Definition 4.1, there exist $\boldsymbol{B} \in \mathbb{R}^{d_{\mathrm{out}},r}$ and $\boldsymbol{A} \in \mathbb{R}^{r,d_{\mathrm{in}}}$ such that

$$\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A} \tag{21}$$

. Let the column vectors of $\boldsymbol{B}$ be $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r$ and the column vectors of $\boldsymbol{W}$ be $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d_{\mathrm{in}}}$. If we write the j-th column of $\boldsymbol{A}$ as $\boldsymbol{a}^{(j)} \in \mathbb{R}^r$, then from equation (21), for each $j \in [1, d_{\mathrm{in}}]_{\mathbb{Z}}$,

$$\boldsymbol{w}_j = \boldsymbol{B}\boldsymbol{a}^{(j)} = \sum_{i=1}^{r} a_i^{(j)} \boldsymbol{b}_i \tag{22}$$

. Therefore, any column vector $\boldsymbol{w}_j$ can be expressed as a linear combination of $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\}$, so

$$\mathcal{C}(\boldsymbol{W}) \subset \mathrm{span}\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\} \tag{23}$$

holds. Thus,

$$\dim \mathcal{C}(\boldsymbol{W}) \leq r = \mathrm{rank}(\boldsymbol{W}) \tag{24}$$

.

To show the reverse inequality, let $s \coloneqq \dim \mathcal{C}(\boldsymbol{W})$ and let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_s$ be a basis for $\mathcal{C}(\boldsymbol{W})$. Define the matrix $\boldsymbol{B}' \in \mathbb{R}^{d_{\mathrm{out}},s}$ having these as columns:

$$\boldsymbol{B}' \coloneqq \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_s \end{bmatrix} \tag{25}$$

. Since each column vector $\boldsymbol{w}_j$ is an element of the column space, there exists a coefficient vector $\boldsymbol{c}^{(j)} \in \mathbb{R}^s$ such that

$$\boldsymbol{w}_j = \sum_{i=1}^{s} c_i^{(j)} \boldsymbol{u}_i \tag{26}$$

. By arranging these coefficients into columns,

$$\boldsymbol{A}' \coloneqq \begin{bmatrix} \boldsymbol{c}^{(1)} & \boldsymbol{c}^{(2)} & \cdots & \boldsymbol{c}^{(d_{\text{in}})} \end{bmatrix} \in \mathbb{R}^{s,d_{\text{in}}} \tag{27}$$

, it follows from equations (25) and (27) that

$$\boldsymbol{W} = \boldsymbol{B}'\boldsymbol{A}' \tag{28}$$

. That is, $s \in S(\boldsymbol{W})$, and by Definition 4.1,

$$\operatorname{rank}(\boldsymbol{W}) \leq s = \dim C(\boldsymbol{W}) \tag{29}$$

holds. Combining equations (24) and (29),

$$\operatorname{rank}(\boldsymbol{W}) = \dim C(\boldsymbol{W}) \tag{30}$$

follows. This proves (1).

Next, we show (2). First, from Definition 4.1, we show that for any $\boldsymbol{W}$,

$$\operatorname{rank}(\boldsymbol{W}) = \operatorname{rank}(\boldsymbol{W}^\top) \tag{31}$$

holds. In fact, if $r \in S(\boldsymbol{W})$, then there exist $\boldsymbol{B}, \boldsymbol{A}$ such that equation (21) holds. Taking the transpose,

$$\boldsymbol{W}^\top = \boldsymbol{A}^\top \boldsymbol{B}^\top \tag{32}$$

. Here, $\boldsymbol{A}^\top \in \mathbb{R}^{d_{\text{in}},r}$ and $\boldsymbol{B}^\top \in \mathbb{R}^{r,d_{\text{out}}}$, so $r \in S(\boldsymbol{W}^\top)$. The reverse can be shown similarly, so $S(\boldsymbol{W}) = S(\boldsymbol{W}^\top)$, and thus equation (31) holds.

On the other hand, the row space $\mathcal{R}(\boldsymbol{W})$ is the subspace spanned by the row vectors of $\boldsymbol{W}$, which is identical to the column space $C(\boldsymbol{W}^\top)$ of $\boldsymbol{W}^\top$. Therefore,

$$\mathcal{R}(\boldsymbol{W}) = C(\boldsymbol{W}^\top) \tag{33}$$

. Applying (1) to $\boldsymbol{W}^\top$,

$$\operatorname{rank}(\boldsymbol{W}^\top) = \dim C(\boldsymbol{W}^\top) \tag{34}$$

, and combining equations (31), (33), and (34), we get

$$\operatorname{rank}(\boldsymbol{W}) = \operatorname{rank}(\boldsymbol{W}^\top) = \dim C(\boldsymbol{W}^\top) = \dim \mathcal{R}(\boldsymbol{W}) \tag{35}$$

. This shows (2) and completes the proof of the proposition. □

Remark 4.1. From Proposition 4.1, $\operatorname{rank}(\boldsymbol{W}) = r$ is equivalent to the dimension of the column space of $\boldsymbol{W}$ being r, i.e., "the number of linearly independent column vectors needed to generate all column vectors of $\boldsymbol{W}$ is r". On the other hand, according to Definition 4.1, when

$\mathrm{rank}(\boldsymbol{W}) = r$, $\boldsymbol{W}$ can be expressed using $\boldsymbol{B} \in \mathbb{R}^{d_{\mathrm{out}}, r}$ and $\boldsymbol{A} \in \mathbb{R}^{r, d_{\mathrm{in}}}$ as in equation (21). In this case, the number of elements in $\boldsymbol{B}$ is $d_{\mathrm{out}} r$, and the number of elements in $\boldsymbol{A}$ is $r d_{\mathrm{in}}$, so $\boldsymbol{W}$ can be described by specifying a total of

$$d_{\mathrm{rank}} \coloneqq d_{\mathrm{out}} r + r d_{\mathrm{in}} \tag{36}$$

scalars. In particular, when $r \ll \min\{d_{\mathrm{out}}, d_{\mathrm{in}}\}$,

$$d_{\mathrm{rank}} \ll d_{\mathrm{out}} d_{\mathrm{in}} \tag{37}$$

, so a low-rank matrix is a class of matrices that can be described with far fewer parameters compared to handling the original full-rank matrix directly. This is perfectly consistent with the motivation of PEFT, which is to represent models and differences as much as possible with a small number of parameters.

LoRA improves parameter efficiency by artificially constraining the difference matrix $\Delta \boldsymbol{W}$ to be represented in this form.

4.2 Parameterization of the Difference Matrix in LoRA

In LoRA, the difference matrix $\Delta \boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}}, d_{\mathrm{in}}}$ for a matrix parameter $\boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}}, d_{\mathrm{in}}}$ is parameterized as follows.

Definition 4.2 (Difference Matrix in LoRA)**.** Fix $d_{\mathrm{out}}, d_{\mathrm{in}} \in \mathbb{Z}_{>0}$, and let the original matrix parameter be $\boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}}, d_{\mathrm{in}}}$. Let $r \in \mathbb{Z}_{>0}$ be a parameter called **rank**, and $\alpha \in \mathbb{R}$ be a parameter called **scaling factor**.
At this time, the difference matrix $\Delta \boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}}, d_{\mathrm{in}}}$ by LoRA is defined as

$$\Delta \boldsymbol{W} \coloneqq \frac{\alpha}{r} \boldsymbol{B} \boldsymbol{A} \tag{38}$$

. Here,

$$\boldsymbol{A} \in \mathbb{R}^{r, d_{\mathrm{in}}}, \quad \boldsymbol{B} \in \mathbb{R}^{d_{\mathrm{out}}, r} \tag{39}$$

are matrices determined by learning, and are the main learning targets of LoRA.
The matrix parameter after applying LoRA is defined as

$$\widetilde{\boldsymbol{W}} \coloneqq \boldsymbol{W} + \Delta \boldsymbol{W} = \boldsymbol{W} + \frac{\alpha}{r} \boldsymbol{B} \boldsymbol{A} \tag{40}$$

.

Remark 4.2. As an implementation note regarding LoRA, the α / r appearing in equation (38) is redundant in a strict sense. That is, the same expressive power can be maintained by setting α to 1 and incorporating the scalar into $\boldsymbol{A}$ or $\boldsymbol{B}$.
In reality, the LoRA paper [1], after including α and r in the formulation as above, explicitly

states that α is set equal to the rank r, i.e., $\alpha = r$, as an experimental setup. In this case, the coefficient α/r in equation (38) is always 1, which is mathematically equivalent to adding $\Delta \boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$. The inclusion of α and r in the formulation is for the convenience of theoretical explanation.
Also, in the LoRA training script `train_text_to_image_lora.py` for Stable Diffusion in Hugging Face diffusers, the setting for LoRA added to the UNet is `LoraConfig(r=args.rank, lora_alpha=args.rank, ...)`, which is designed such that $\alpha = r$ in implementation as well[a]. Therefore, although α and r are mathematically redundant, the form $\frac{\alpha}{r}$ is preserved in both the original paper and representative implementations, and on top of that, it is typically used with $\alpha = r$.

[a] See the code example in the Hugging Face diffusers LoRA training guide. At `https://huggingface.co/docs/diffusers/training/lora`, the setting `unet_lora_config = LoraConfig(r=args.rank, lora_alpha=args.rank, init_lora_weights="gaussian", target_modules=["to_k", "to_q", "to_v", "to_out.0"])` is shown, from which it is clear that $\alpha = r$.

Let's count the number of parameters in the difference matrix. Whereas the original full degrees of freedom are $d_{\text{out}}d_{\text{in}}$, in LoRA, the degrees of freedom for $\boldsymbol{A}, \boldsymbol{B}$ are

$$d_{\text{LoRA}} \coloneqq r d_{\text{in}} + d_{\text{out}} r = r(d_{\text{in}} + d_{\text{out}}) \tag{41}$$

and one for the scalar α, totaling $r(d_{\text{in}} + d_{\text{out}}) + 1$. Usually, we choose $r \ll \min\{d_{\text{in}}, d_{\text{out}}\}$, so $d_{\text{LoRA}} \ll d_{\text{out}}d_{\text{in}}$.

Example 4.1 (Comparison of Specific Parameter Counts)**.** As specific numbers, consider the case $d_{\text{in}} = 768$, $d_{\text{out}} = 3072$, $r = 32$. This is a size typically appearing in the linear transformations of intermediate layers in Transformer-based models [3, 4].
First, the number of elements in the original matrix parameter $\boldsymbol{W} \in \mathbb{R}^{3072,768}$ is

$$d_{\text{full}} = d_{\text{out}}d_{\text{in}} = 3072 \times 768 \tag{42}$$

. Calculating this gives

$$d_{\text{full}} = 3072 \times 768 = 2,359,296 \tag{43}$$

scalars.
On the other hand, the number of elements in $\boldsymbol{A}, \boldsymbol{B}$ representing the difference by LoRA is

$$d_{\text{LoRA}} = r(d_{\text{in}} + d_{\text{out}}) = 32(768 + 3072) = 32 \times 3840 = 122,880 \tag{44}$$

, and even adding the scalar α, it is $122,881$.
Therefore, compared to representing the difference matrix at full size, the number of parameters required is

$$\frac{d_{\text{LoRA}}}{d_{\text{full}}} \approx \frac{1.23 \times 10^5}{2.36 \times 10^6} \approx 0.052 \tag{45}$$

, which is reduced to about $1/19$. This is for representing the difference for a single matrix,

and in practice, LoRA is applied to many matrices in the network, but a similar reduction effect is obtained for each matrix.

4.3 General Definition of LoRA

Next, we define the framework for applying LoRA to a general neural network containing multiple matrix parameters.

Definition 4.3 (LoRA for General Neural Networks)**.** Let $f_{\boldsymbol{W}_1,\boldsymbol{W}_2,\ldots,\boldsymbol{W}_K,\boldsymbol{\theta}_{\text{others}}}$ be a neural network consisting of K matrix parameters and other parameters. That is, $\boldsymbol{W}_k \in \mathbb{R}^{d_{\text{out},k},d_{\text{in},k}}$ $(k = 1,\ldots,K)$ and

$$\boldsymbol{\theta}_{\text{others}} \in \mathbb{R}^{d_{\text{others}}} \tag{46}$$

. Let the entire set of parameters be

$$\Theta = \left(\boldsymbol{W}_1,\ldots,\boldsymbol{W}_K,\boldsymbol{\theta}_{\text{others}}\right) \tag{47}$$

, and this is the parameter of the model f.
For each $k \in [1,K]_{\mathbb{Z}}$, fix a rank $r_k \in \mathbb{Z}_{>0}$, and let

$$\boldsymbol{A}_k \in \mathbb{R}^{r_k,d_{\text{in},k}}, \quad \boldsymbol{B}_k \in \mathbb{R}^{d_{\text{out},k},r_k} \tag{48}$$

be the LoRA matrix parameters, and $\alpha_k \in \mathbb{R}$ be the LoRA scaling factor. Also, let $\eta \in \mathbb{R}$ be a parameter that controls the overall scaling of LoRA. η is usually a hyperparameter set at inference time and is often not a learning target.
At this time, the **LoRA adapter** is defined as

$$\begin{aligned}&\text{LoRA}_{f_{(\cdot)},\eta,\alpha_1,\boldsymbol{A}_1,\boldsymbol{B}_1,\ldots,\alpha_K,\boldsymbol{A}_K,\boldsymbol{B}_K}(\boldsymbol{W}_1,\ldots,\boldsymbol{W}_K,\boldsymbol{\theta}_{\text{others}})\\ &\quad := f_{\boldsymbol{W}_1+\eta\frac{\alpha_1}{r_1}\boldsymbol{B}_1\boldsymbol{A}_1,\boldsymbol{W}_2+\eta\frac{\alpha_2}{r_2}\boldsymbol{B}_2\boldsymbol{A}_2,\ldots,\boldsymbol{W}_K+\eta\frac{\alpha_K}{r_K}\boldsymbol{B}_K\boldsymbol{A}_K,\boldsymbol{\theta}_{\text{others}}}.\end{aligned} \tag{49}$$

Here, for $k = 1,2,\ldots,K$, r_k is a predetermined positive integer and is not changed during learning. Also, α_k is usually a fixed positive number.

Remark 4.3. In Definition 4.3, η is a parameter that adjusts the "strength" of LoRA. In the learning phase, learning is often performed with $\eta = 1$, and by changing η in the inference phase,

$$\widetilde{\boldsymbol{W}}_k(\eta) = \boldsymbol{W}_k + \eta\frac{\alpha_k}{r_k}\boldsymbol{B}_k\boldsymbol{A}_k \tag{50}$$

, the contribution of the modification by LoRA can be strengthened or weakened. Setting $\eta = 0$ returns to the original model $f_{\boldsymbol{W}_1,\ldots,\boldsymbol{W}_K,\boldsymbol{\theta}_{\text{others}}}$.

4.4 Training LoRA Parameters

Under Definition 4.3, training LoRA means determining $\boldsymbol{A}_k, \boldsymbol{B}_k$ $(k = 1, \ldots, K)$ based on the training data. Here, we assume that the original matrix parameters $\boldsymbol{W}_k$ and other parameters $\boldsymbol{\theta}_{\text{others}}$ are fixed and not updated.

Given m input-output pairs $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ for $i = 1, \ldots, m$, define the objective function (loss function) as

$$\mathcal{L}(\alpha_1, \boldsymbol{A}_1, \boldsymbol{B}_1, \ldots, \alpha_K, \boldsymbol{A}_K, \boldsymbol{B}_K) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \ell\big(\boldsymbol{y}^{(i)}, f^{\text{LoRA}}(\boldsymbol{x}^{(i)})\big) \tag{51}$$

. Here,

$$f^{\text{LoRA}}(\boldsymbol{x}) \coloneqq \text{LoRA}_{f_{(\cdot)}, \eta=1, \alpha_1, \boldsymbol{A}_1, \boldsymbol{B}_1, \ldots, \alpha_K, \boldsymbol{A}_K, \boldsymbol{B}_K}(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_K, \boldsymbol{\theta}_{\text{others}})(\boldsymbol{x}) \tag{52}$$

$$= f_{\boldsymbol{W}_1 + \frac{\alpha_1}{r_1}\boldsymbol{B}_1\boldsymbol{A}_1, \ldots, \boldsymbol{W}_K + \frac{\alpha_K}{r_K}\boldsymbol{B}_K\boldsymbol{A}_K, \boldsymbol{\theta}_{\text{others}}}(\boldsymbol{x}) \tag{53}$$

, and ℓ is a task-dependent loss function, such as mean squared error or cross-entropy.

Definition 4.4 (LoRA Training Problem)**.** The LoRA training problem is, given the fixed original parameters $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_K, \boldsymbol{\theta}_{\text{others}}$, to solve

$$\underset{\boldsymbol{A}_1, \boldsymbol{B}_1, \ldots, \boldsymbol{A}_K, \boldsymbol{B}_K}{\text{Minimize}} \quad \mathcal{L}(\boldsymbol{A}_1, \boldsymbol{B}_1, \ldots, \boldsymbol{A}_K, \boldsymbol{B}_K) \tag{54}$$

.

Remark 4.4. In actual training, a gradient method (stochastic gradient descent, AdamW, etc.) is used to numerically solve the minimization problem in Definition 4.4. That is, at each step,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}_k}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{B}_k} \tag{55}$$

are obtained by backpropagation, and $\boldsymbol{A}_k, \boldsymbol{B}_k$ are updated based on them. At this time, gradients with respect to $\boldsymbol{W}_k$ and $\boldsymbol{\theta}_{\text{others}}$ are calculated but not used for updates (they are not updated). This allows training only the LoRA parameters without changing the original model.

4.5 Controlling LoRA Strength After Training

After the LoRA parameters $\alpha_k, \boldsymbol{A}_k, \boldsymbol{B}_k$ are determined by Definition 4.4, the strength of the LoRA contribution can be controlled using η from Definition 4.3 in the inference phase.

That is, at inference time, by specifying an arbitrary $\eta \in \mathbb{R}$ and defining

$$f^{\text{LoRA},\eta}(\boldsymbol{x}) = f_{\boldsymbol{W}_1 + \eta\frac{\alpha_1}{r_1}\boldsymbol{B}_1\boldsymbol{A}_1, \ldots, \boldsymbol{W}_K + \eta\frac{\alpha_K}{r_K}\boldsymbol{B}_K\boldsymbol{A}_K, \boldsymbol{\theta}_{\text{others}}}(\boldsymbol{x}) \tag{56}$$

, one can return to the original model with $\eta = 0$, use the same strength as during training

with $\eta = 1$, or strengthen or weaken the LoRA contribution more than usual with $|\eta| > 1$.

In implementation, by allowing the user to explicitly specify η, the "strength of the LoRA flavor" can be easily adjusted. This is frequently used for purposes such as adjusting style transfer or the tendency of prompt responses.

5 Specific Application of LoRA: Example with Stable Diffusion 1.5

In this section, we will outline the application of LoRA in Stable Diffusion 1.5 [5, 6] as a specific example. Here, we will mathematically describe which matrix parameters LoRA is applied to, referring to the layer names in the Python libraries `diffusers`[1] and `transformers`[2].

5.1 Overview of Stable Diffusion 1.5 and LoRA Application Points

Stable Diffusion 1.5 is broadly composed of the following components:

- **Text encoder**: Usually a `CLIPTextModel` from `transformers` or a similar model, which generates embeddings (condition vectors) from the input text token sequence [7].
- **U-Net denoiser** (U-Net style noise estimator): Implemented as `UNet2DConditionModel` in `diffusers`, it estimates noise from a noisy latent variable. It includes self-attention and cross-attention blocks.
- **VAE (variational auto-encoder)**: Implemented as `AutoencoderKL` or similar, it handles the mutual conversion between images and latent representations.

In practice, the main places where LoRA is applied are the following two:

- Targeting only the U-Net: Apply LoRA to the linear layers related to self-attention and cross-attention in the noise estimator (especially `to_q`, `to_k`, `to_v`, `to_out`).
- Also applying to the text encoder: In addition to the above, apply LoRA to the self-attention layers (`self.attn.q_proj`, `k_proj`, `v_proj`, `out_proj`) or MLP parts of the `CLIPTextModel`.

In the following, we will define the functions represented by each layer, separating input variables and parameters, and strictly describe in mathematical formulas which matrices LoRA is applied to.

[1] Official repository for the `diffusers` library: `https://github.com/huggingface/diffusers`
[2] Official repository for the `transformers` library: `https://github.com/huggingface/transformers`

5.2 LoRA in Self-Attention and Cross-Attention within U-Net

The attention blocks in the U-Net of Stable Diffusion 1.5 are often implemented as the `diffusers.models.attention.CrossAttention` class. Here, we will abstractly define the function determined by this attention block.

5.2.1 Mathematical Representation of the CrossAttention Layer

Let d_{model} be the dimension of the feature vectors, d_{head} be the dimension of each head, and n_{head} be the number of heads, such that

$$d_{\mathrm{model}} = n_{\mathrm{head}} d_{\mathrm{head}} \tag{57}$$

.

Definition 5.1 (CrossAttention Layer in U-Net)**.** Let the query input sequence and the key/-value input sequences be

$$\boldsymbol{X}_{\mathrm{q}} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{model}}}, \quad \boldsymbol{X}_{\mathrm{k}} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{model}}}, \quad \boldsymbol{X}_{\mathrm{v}} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{model}}} \tag{58}$$

. Here, $L_{\mathrm{q}}, L_{\mathrm{k}}$ are the query sequence length and the key/value sequence length, respectively.
The CrossAttention layer of the U-Net has the following matrix parameters:

$$\boldsymbol{W}_{\mathrm{q}} \in \mathbb{R}^{d_{\mathrm{model}}, d_{\mathrm{model}}}, \tag{59}$$
$$\boldsymbol{W}_{\mathrm{k}} \in \mathbb{R}^{d_{\mathrm{model}}, d_{\mathrm{model}}}, \tag{60}$$
$$\boldsymbol{W}_{\mathrm{v}} \in \mathbb{R}^{d_{\mathrm{model}}, d_{\mathrm{model}}}, \tag{61}$$
$$\boldsymbol{W}_{\mathrm{o}} \in \mathbb{R}^{d_{\mathrm{model}}, d_{\mathrm{model}}}. \tag{62}$$

These correspond to `to_q`, `to_k`, `to_v`, `to_out` in the diffusers implementation.
First, by linear transformation,

$$\boldsymbol{Q} := \boldsymbol{X}_{\mathrm{q}} \boldsymbol{W}_{\mathrm{q}}^{\top} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{model}}}, \tag{63}$$
$$\boldsymbol{K} := \boldsymbol{X}_{\mathrm{k}} \boldsymbol{W}_{\mathrm{k}}^{\top} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{model}}}, \tag{64}$$
$$\boldsymbol{V} := \boldsymbol{X}_{\mathrm{v}} \boldsymbol{W}_{\mathrm{v}}^{\top} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{model}}} \tag{65}$$

are determined. $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are split for each head, and written as

$$\boldsymbol{Q} \leftrightarrow \left(\boldsymbol{Q}^{(h)}\right)_{h=1}^{n_{\mathrm{head}}}, \quad \boldsymbol{Q}^{(h)} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{head}}}, \tag{66}$$
$$\boldsymbol{K} \leftrightarrow \left(\boldsymbol{K}^{(h)}\right)_{h=1}^{n_{\mathrm{head}}}, \quad \boldsymbol{K}^{(h)} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{head}}}, \tag{67}$$
$$\boldsymbol{V} \leftrightarrow \left(\boldsymbol{V}^{(h)}\right)_{h=1}^{n_{\mathrm{head}}}, \quad \boldsymbol{V}^{(h)} \in \mathbb{R}^{L_{\mathrm{k}}, d_{\mathrm{head}}} \tag{68}$$

. Scaled dot-product attention is calculated for each head. That is,

$$\boldsymbol{S}^{(h)} \coloneqq \frac{1}{\sqrt{d_{\mathrm{head}}}} \boldsymbol{Q}^{(h)} \boldsymbol{K}^{(h)\top} \in \mathbb{R}^{L_{\mathrm{q}}, L_{\mathrm{k}}} \tag{69}$$

, and applying softmax row-wise,

$$\boldsymbol{A}^{(h)} \coloneqq \mathrm{softmax}_{\mathrm{row}}(\boldsymbol{S}^{(h)}) \in \mathbb{R}^{L_{\mathrm{q}}, L_{\mathrm{k}}} \tag{70}$$

is determined. Then

$$\boldsymbol{O}^{(h)} \coloneqq \boldsymbol{A}^{(h)} \boldsymbol{V}^{(h)} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{head}}} \tag{71}$$

is obtained. The outputs for $h = 1, \ldots, n_{\mathrm{head}}$ are concatenated in the feature dimension direction

$$\boldsymbol{O} \coloneqq \mathrm{Concat}_{h=1}^{n_{\mathrm{head}}} \boldsymbol{O}^{(h)} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{model}}} \tag{72}$$

, and finally

$$\boldsymbol{Y} \coloneqq \boldsymbol{O} \boldsymbol{W}_{\mathrm{o}}^{\top} \in \mathbb{R}^{L_{\mathrm{q}}, d_{\mathrm{model}}} \tag{73}$$

is the output of the CrossAttention layer.

Remark 5.1. Definition 5.1 writes down the standard multi-head attention of the Transformer [3], adapted for the attention blocks of the U-Net. Although batch dimensions, positional encodings, etc., are included in the implementation, the important point is that the linear layers (63)–(65), (73) targeted by LoRA have been explicitly shown in mathematical formulas.

5.2.2 Applying LoRA to the CrossAttention Layer

A typical method for applying LoRA is to add separate LoRAs to each of $\boldsymbol{W}_{\mathrm{q}}, \boldsymbol{W}_{\mathrm{k}}, \boldsymbol{W}_{\mathrm{v}}, \boldsymbol{W}_{\mathrm{o}}$. For example, consider LoRA for $\boldsymbol{W}_{\mathrm{q}}$.

Prepare rank r_{q}, scaling factor α_{q}, matrices $\boldsymbol{A}_{\mathrm{q}} \in \mathbb{R}^{r_{\mathrm{q}}, d_{\mathrm{model}}}$, $\boldsymbol{B}_{\mathrm{q}} \in \mathbb{R}^{d_{\mathrm{model}}, r_{\mathrm{q}}}$, and

$$\widetilde{\boldsymbol{W}}_{\mathrm{q}} \coloneqq \boldsymbol{W}_{\mathrm{q}} + \eta \frac{\alpha_{\mathrm{q}}}{r_{\mathrm{q}}} \boldsymbol{B}_{\mathrm{q}} \boldsymbol{A}_{\mathrm{q}} \tag{74}$$

is determined. Similarly,

$$\widetilde{\boldsymbol{W}}_{\mathrm{k}} \coloneqq \boldsymbol{W}_{\mathrm{k}} + \eta \frac{\alpha_{\mathrm{k}}}{r_{\mathrm{k}}} \boldsymbol{B}_{\mathrm{k}} \boldsymbol{A}_{\mathrm{k}}, \tag{75}$$

$$\widetilde{\boldsymbol{W}}_{\mathrm{v}} \coloneqq \boldsymbol{W}_{\mathrm{v}} + \eta \frac{\alpha_{\mathrm{v}}}{r_{\mathrm{v}}} \boldsymbol{B}_{\mathrm{v}} \boldsymbol{A}_{\mathrm{v}}, \tag{76}$$

$$\widetilde{\boldsymbol{W}}_{\mathrm{o}} \coloneqq \boldsymbol{W}_{\mathrm{o}} + \eta \frac{\alpha_{\mathrm{o}}}{r_{\mathrm{o}}} \boldsymbol{B}_{\mathrm{o}} \boldsymbol{A}_{\mathrm{o}} \tag{77}$$

are defined. Here, $r_{\mathrm{q}}, r_{\mathrm{k}}, r_{\mathrm{v}}, r_{\mathrm{o}}$ are the ranks for each respective matrix, and in implementation, a common value is often used.

The CrossAttention layer after applying LoRA is defined as in Definition 5.1, but replacing

$\boldsymbol{W}_{\mathrm{q}}, \boldsymbol{W}_{\mathrm{k}}, \boldsymbol{W}_{\mathrm{v}}, \boldsymbol{W}_{\mathrm{o}}$ with $\widetilde{\boldsymbol{W}}_{\mathrm{q}}, \widetilde{\boldsymbol{W}}_{\mathrm{k}}, \widetilde{\boldsymbol{W}}_{\mathrm{v}}, \widetilde{\boldsymbol{W}}_{\mathrm{o}}$ respectively in (63)–(73).

5.3 LoRA in the Text Encoder

The text encoder of Stable Diffusion 1.5 is the CLIP text model [7], implemented as `CLIPTextModel` in `transformers`. Here too, LoRA is often applied to the linear transformations of the self-attention layers.

5.3.1 Mathematical Representation of Self-Attention in CLIPTextModel

Let $d_{\mathrm{model}}^{\mathrm{text}}$ be the dimension of the text embedding, $n_{\mathrm{head}}^{\mathrm{text}}$ be the number of heads, and $d_{\mathrm{head}}^{\mathrm{text}}$ be the dimension of each head.

$$d_{\mathrm{model}}^{\mathrm{text}} = n_{\mathrm{head}}^{\mathrm{text}} d_{\mathrm{head}}^{\mathrm{text}} \tag{78}$$

. Let the input sequence be

$$\boldsymbol{X}_{\mathrm{text}} \in \mathbb{R}^{L_{\mathrm{text}}, d_{\mathrm{model}}^{\mathrm{text}}} \tag{79}$$

.

Definition 5.2 (CLIP Text Self-Attention Layer)**.** The self-attention layer of the CLIP text model has the following matrix parameters:

$$\boldsymbol{W}_{\mathrm{q}}^{\mathrm{text}} \in \mathbb{R}^{d_{\mathrm{model}}^{\mathrm{text}}, d_{\mathrm{model}}^{\mathrm{text}}}, \tag{80}$$
$$\boldsymbol{W}_{\mathrm{k}}^{\mathrm{text}} \in \mathbb{R}^{d_{\mathrm{model}}^{\mathrm{text}}, d_{\mathrm{model}}^{\mathrm{text}}}, \tag{81}$$
$$\boldsymbol{W}_{\mathrm{v}}^{\mathrm{text}} \in \mathbb{R}^{d_{\mathrm{model}}^{\mathrm{text}}, d_{\mathrm{model}}^{\mathrm{text}}}, \tag{82}$$
$$\boldsymbol{W}_{\mathrm{o}}^{\mathrm{text}} \in \mathbb{R}^{d_{\mathrm{model}}^{\mathrm{text}}, d_{\mathrm{model}}^{\mathrm{text}}}. \tag{83}$$

Query, key, and value are defined as

$$\boldsymbol{Q}^{\mathrm{text}} \coloneqq \boldsymbol{X}_{\mathrm{text}} \boldsymbol{W}_{\mathrm{q}}^{\mathrm{text}\top}, \tag{84}$$
$$\boldsymbol{K}^{\mathrm{text}} \coloneqq \boldsymbol{X}_{\mathrm{text}} \boldsymbol{W}_{\mathrm{k}}^{\mathrm{text}\top}, \tag{85}$$
$$\boldsymbol{V}^{\mathrm{text}} \coloneqq \boldsymbol{X}_{\mathrm{text}} \boldsymbol{W}_{\mathrm{v}}^{\mathrm{text}\top}, \tag{86}$$

, and then multi-head self-attention is calculated similarly to Definition 5.1, and the output is

$$\boldsymbol{Y}^{\mathrm{text}} \coloneqq \boldsymbol{O}^{\mathrm{text}} \boldsymbol{W}_{\mathrm{o}}^{\mathrm{text}\top} \tag{87}$$

.

5.3.2 LoRA for CLIP Text Self-Attention

The application of LoRA is similar to the CrossAttention in the U-Net, and for each matrix parameter,

$$\widetilde{\boldsymbol{W}}_{\mathrm{q}}^{\mathrm{text}} = \boldsymbol{W}_{\mathrm{q}}^{\mathrm{text}} + \eta \frac{\alpha_{\mathrm{q}}^{\mathrm{text}}}{r_{\mathrm{q}}^{\mathrm{text}}} \boldsymbol{B}_{\mathrm{q}}^{\mathrm{text}} \boldsymbol{A}_{\mathrm{q}}^{\mathrm{text}}, \tag{88}$$

$$\widetilde{\boldsymbol{W}}_{\mathrm{k}}^{\mathrm{text}} = \boldsymbol{W}_{\mathrm{k}}^{\mathrm{text}} + \eta \frac{\alpha_{\mathrm{k}}^{\mathrm{text}}}{r_{\mathrm{k}}^{\mathrm{text}}} \boldsymbol{B}_{\mathrm{k}}^{\mathrm{text}} \boldsymbol{A}_{\mathrm{k}}^{\mathrm{text}}, \tag{89}$$

$$\widetilde{\boldsymbol{W}}_{\mathrm{v}}^{\mathrm{text}} = \boldsymbol{W}_{\mathrm{v}}^{\mathrm{text}} + \eta \frac{\alpha_{\mathrm{v}}^{\mathrm{text}}}{r_{\mathrm{v}}^{\mathrm{text}}} \boldsymbol{B}_{\mathrm{v}}^{\mathrm{text}} \boldsymbol{A}_{\mathrm{v}}^{\mathrm{text}}, \tag{90}$$

$$\widetilde{\boldsymbol{W}}_{\mathrm{o}}^{\mathrm{text}} = \boldsymbol{W}_{\mathrm{o}}^{\mathrm{text}} + \eta \frac{\alpha_{\mathrm{o}}^{\mathrm{text}}}{r_{\mathrm{o}}^{\mathrm{text}}} \boldsymbol{B}_{\mathrm{o}}^{\mathrm{text}} \boldsymbol{A}_{\mathrm{o}}^{\mathrm{text}} \tag{91}$$

are determined. This modifies the self-attention of the text encoder by LoRA. In practice, by combining the LoRA of the text encoder with the LoRA of the U-Net, both the tendency of prompt interpretation and the image generation process are controlled.

5.4 Implementation Handling of LoRA at Inference Time

Finally, we describe two typical methods for handling LoRA in implementation at inference time.

Remark 5.2 (Fusing LoRA vs. On-the-fly Application)**.** There are at least the following two options for how to specifically use the LoRA-applied matrix $\widetilde{\boldsymbol{W}} = \boldsymbol{W} + \frac{\alpha}{r}\boldsymbol{B}\boldsymbol{A}$ in calculations.

- **Pre-fusing method**: Before performing inference, $\boldsymbol{W}_{\mathrm{LoRA}} \coloneqq \boldsymbol{W} + \frac{\alpha}{r}\boldsymbol{B}\boldsymbol{A}$ is calculated once and saved, and thereafter $\boldsymbol{W}_{\mathrm{LoRA}}$ is always used to calculate $\boldsymbol{W}_{\mathrm{LoRA}}\boldsymbol{x}$. In the `diffusers` library, this is implemented as `fuse_lora()`. This method is advantageous in terms of computation, because once $\boldsymbol{W}_{\mathrm{LoRA}}$ is calculated, there is no need to calculate $\boldsymbol{B}\boldsymbol{A}\boldsymbol{x}$ every time during numerous inferences (image generation steps).
- **Applying LoRA during each forward pass method**: Even at inference time, the original matrix $\boldsymbol{W}$ is kept as is, and for the input $\boldsymbol{x}$,

  $$\boldsymbol{W}\boldsymbol{x} + \frac{\alpha}{r}\boldsymbol{B}\boldsymbol{A}\boldsymbol{x} \tag{92}$$

  is calculated every time. This method is highly flexible when one wants to switch multiple LoRAs at once or turn LoRA on/off during inference. It is also useful in situations where one absolutely does not want to change the original $\boldsymbol{W}$.

In general, the on-the-fly method is convenient in situations where LoRA application and removal are frequently repeated, while the fuse method is advantageous in terms of computational efficiency when a certain LoRA is fixed for a long time to perform a large amount of inference.

6 Summary and Next Lecture Preview

6.1 Answers to Learning Outcomes

We briefly review the Learning Outcomes set forth in this lecture.

- LoRA is a type of parameter-efficient fine-tuning (PEFT) that can efficiently realize a better model for solving a task by determining a small number of parameters that represent the difference from the original model.
- LoRA requires changing only a small number of parameters, yet possesses sufficient expressive power for applications. This brings practical advantages such as improved learning stability with small data, reduction in storage usage, and ease of managing numerous task-specific adapters.
- Mathematically, LoRA restricts the difference matrix to one represented by the product of a matrix with few rows and a matrix with few columns, and determines those matrices with few rows and few columns.

6.2 Next Lecture Preview

Next time, we will explain the **relationship between the number of parameters or the size of the function space, and the amount of data required for learning**. That is, we will outline the relationship between model capacity, generalization performance, and the required sample size from the perspective of statistics and learning theory, and consider how methods like PEFT and LoRA can contribute to data efficiency.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," arXiv preprint arXiv:2106.09685, 2021.

[2] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," Advances in Neural Information Processing Systems, 2023. NeurIPS 2023.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in Neural Information Processing Systems, vol. 30, 2017.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Technical Report, 2019. Technical report.

[5] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10684–10695, 2022.

[6] S. AI, "Stable diffusion v1-5 model card." Technical report, 2022. Model card and weights.

[7] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," Proceedings of the 38th International Conference on Machine Learning, pp. 8748–8763, 2021.