appendix E
Parameter-efficient fine-tuning with LoRA

Low-rank adaptation (*LoRA*) is one of the most widely used techniques for *parameter-efficient fine-tuning*. The following discussion is based on the spam classification fine-tuning example given in chapter 6. However, LoRA fine-tuning is also applicable to the supervised *instruction fine-tuning* discussed in chapter 7.

E.1 Introduction to LoRA

LoRA is a technique that adapts a pretrained model to better suit a specific, often smaller dataset by adjusting only a small subset of the model's weight parameters. The "low-rank" aspect refers to the mathematical concept of limiting model adjustments to a smaller dimensional subspace of the total weight parameter space. This effectively captures the most influential directions of the weight parameter changes during training. The LoRA method is useful and popular because it enables efficient fine-tuning of large models on task-specific data, significantly cutting down on the computational costs and resources usually required for fine-tuning.

Suppose a large weight matrix W is associated with a specific layer. LoRA can be applied to all linear layers in an LLM. However, we focus on a single layer for illustration purposes.

When training deep neural networks, during backpropagation, we learn a ΔW matrix, which contains information on how much we want to update the original weight parameters to minimize the loss function during training. Hereafter, I use the term "weight" as shorthand for the model's weight parameters.

In regular training and fine-tuning, the weight update is defined as follows:

$$W_{updated} = W + \Delta W$$

The LoRA method, proposed by Hu et al. (https://arxiv.org/abs/2106.09685), offers a more efficient alternative to computing the weight updates ΔW by learning an approximation of it:

$$\Delta W \approx AB$$

where A and B are two matrices much smaller than W, and AB represents the matrix multiplication product between A and B.

Using LoRA, we can then reformulate the weight update we defined earlier:

$$W_{updated} = W + AB$$

Figure E.1 illustrates the weight update formulas for full fine-tuning and LoRA side by side.

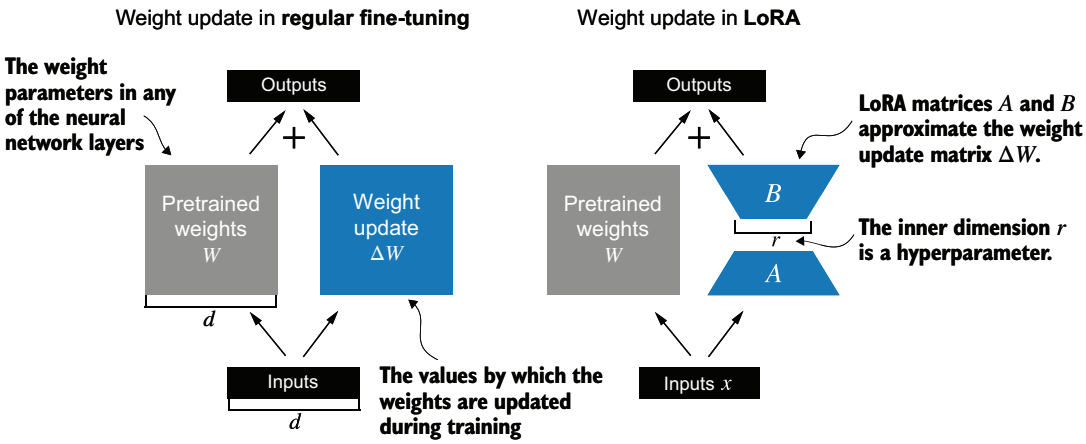

Figure E.1 A comparison between weight update methods: regular fine-tuning and LoRA. Regular fine-tuning involves updating the pretrained weight matrix *W* directly with ΔW (left). LoRA uses two smaller matrices, *A* and *B*, to approximate ΔW, where the product *AB* is added to *W*, and *r* denotes the inner dimension, a tunable hyperparameter (right).

If you paid close attention, you might have noticed that the visual representations of full fine-tuning and LoRA in figure E.1 differ slightly from the earlier presented formulas. This variation is attributed to the distributive law of matrix multiplication, which allows us to separate the original and updated weights rather than combine them. For example, in the case of regular fine-tuning with x as the input data, we can express the computation as

$$x(W + \Delta W) = xW + x\Delta W$$

Similarly, we can write the following for LoRA:

$$x(W + AB) = xW + xAB$$

Besides reducing the number of weights to update during training, the ability to keep the LoRA weight matrices separate from the original model weights makes LoRA even more useful in practice. Practically, this allows for the pretrained model weights to remain unchanged, with the LoRA matrices being applied dynamically after training when using the model.

Keeping the LoRA weights separate is very useful in practice because it enables model customization without needing to store multiple complete versions of an LLM. This reduces storage requirements and improves scalability, as only the smaller LoRA matrices need to be adjusted and saved when we customize LLMs for each specific customer or application.

Next, let's see how LoRA can be used to fine-tune an LLM for spam classification, similar to the fine-tuning example in chapter 6.

E.2 Preparing the dataset

Before applying LoRA to the spam classification example, we must load the dataset and pretrained model we will work with. The code here repeats the data preparation from chapter 6. (Instead of repeating the code, we could open and run the chapter 6 notebook and insert the LoRA code from section E.4 there.)

First, we download the dataset and save it as CSV files.

Listing E.1 Downloading and preparing the dataset

```
from pathlib import Path
import pandas as pd
from ch06 import (
    download_and_unzip_spam_data,
    create_balanced_dataset,
    random_split
)

url = \
"https://archive.ics.uci.edu/static/public/228/sms+spam+collection.zip"
zip_path = "sms_spam_collection.zip"
extracted_path = "sms_spam_collection"
data_file_path = Path(extracted_path) / "SMSSpamCollection.tsv"

download_and_unzip_spam_data(url, zip_path, extracted_path, data_file_path)

df = pd.read_csv(
    data_file_path, sep="\t", header=None, names=["Label", "Text"]
)
balanced_df = create_balanced_dataset(df)
balanced_df["Label"] = balanced_df["Label"].map({"ham": 0, "spam": 1})

train_df, validation_df, test_df = random_split(balanced_df, 0.7, 0.1)
train_df.to_csv("train.csv", index=None)
```