

## *Chapter 2*

# *Efficient Computation*

From an implementation standpoint, deep learning is about executing heavy computations with large amounts of data. The Graphical Processing Units (GPUs) have been instrumental in the success of the field by allowing such computations to be run on affordable hardware.

The importance of their use, and the resulting technical constraints on the computations that can be done efficiently, force the research in the field to constantly balance mathematical soundness and implementability of novel methods.

## 2.1 *GPUs, TPUs, and batches*

Graphical Processing Units were originally designed for real-time image synthesis, which requires highly parallel architectures that happen to be well suited for deep models. As their usage for AI has increased, GPUs have been equipped with dedicated tensor cores, and deep-learning specialized chips such as Google's Tensor Processing Units (TPUs) have been developed.

A GPU possesses several thousand parallel units and its own fast memory. The limiting factor is usually not the number of computing units, but the read-write operations to memory. The slowest link is between the CPU memory and the GPU memory, and consequently one should avoid copying data across devices. Moreover, the structure of the GPU itself involves multiple levels of cache memory, which are smaller but faster, and computation should be organized to avoid copies between these different caches.

This is achieved, in particular, by organizing the computation in batches of samples that can fit entirely in the GPU memory and are processed in parallel. When an operator combines a sample and model parameters, both have to be moved to the cache memory near the actual computing

units. Proceeding by batches allows for copying the model parameters only once, instead of doing it for each sample. In practice, a GPU processes a batch that fits in memory almost as quickly as it would process a single sample.

A standard GPU has a theoretical peak performance of  $10^{13}$ – $10^{14}$  floating-point operations (FLOPs) per second, and its memory typically ranges from 8 to 80 gigabytes. The standard FP32 encoding of float numbers is on 32 bits, but empirical results show that using encoding on 16 bits, or even less for some operands, does not degrade performance.

We will come back in § 3.7 to the large size of deep architectures.

## 2.2 Tensors

GPUs and deep learning frameworks such as PyTorch or JAX manipulate the quantities to be processed by organizing them as tensors, which are series of scalars arranged along several discrete axes. They are elements of  $\mathbb{R}^{N_1 \times \dots \times N_D}$  that generalize the notion of vector and matrix.

Tensors are used to represent both the signals to be processed, the trainable parameters of the models, and the intermediate quantities they compute. The latter are called activations, in reference to neuronal activations.

For instance, a time series is naturally encoded as a  $T \times D$  tensor, or, for historical reasons, as a  $D \times T$  tensor, where  $T$  is its duration and  $D$  is the dimension of the feature representation at every time step, often referred to as the number of channels. Similarly, a 2D-structured signal can be represented as a  $D \times H \times W$  tensor, where  $H$  and  $W$  are its height and width. An RGB image would correspond to  $D = 3$ , but the number of channels can grow up to several thousands in large models.

Adding more dimensions allows for the representation of series of objects. For example, fifty RGB images of resolution  $32 \times 24$  can be encoded as

a  $50 \times 3 \times 24 \times 32$  tensor.

Deep learning libraries provide a large number of operations that encompass standard linear algebra, complex reshaping and extraction, and deep-learning specific operations, some of which we will see in Chapter 4. The implementation of tensors separates the shape representation from the storage layout of the coefficients in memory, which allows many reshaping, transposing, and extraction operations to be done without coefficient copying, hence extremely rapidly.

In practice, virtually any computation can be decomposed into elementary tensor operations, which avoids non-parallel loops at the language level and poor memory management.

Besides being convenient tools, tensors are instrumental in achieving computational efficiency. All the people involved in the development of an operational deep model, from the designers of the drivers, libraries, and models to those of the computers and chips, know that the data will be manipulated as tensors. The resulting constraints on locality and block decomposability enable all the actors in this chain to come up with optimal designs.