

I use PyTorch 2.4.0 for the examples, so I recommend that you use the following command to install the exact version to guarantee compatibility with this book:

```
pip install torch==2.4.0
```

However, as mentioned earlier, given your operating system, the installation command might differ slightly from the one shown here. Thus, I recommend that you visit <https://pytorch.org> and use the installation menu (see figure A.4) to select the installation command for your operating system. Remember to replace `torch` with `torch==2.4.0` in the command.

To check the version of PyTorch, execute the following code in PyTorch:

```
import torch
torch.__version__
```

This prints

```
'2.4.0'
```

PyTorch and Torch

The Python library is named PyTorch primarily because it's a continuation of the Torch library but adapted for Python (hence, "PyTorch"). "Torch" acknowledges the library's roots in Torch, a scientific computing framework with wide support for machine learning algorithms, which was initially created using the Lua programming language.

If you are looking for additional recommendations and instructions for setting up your Python environment or installing the other libraries used in this book, visit the supplementary GitHub repository of this book at <https://github.com/rasbt/LLMs-from-scratch>.

After installing PyTorch, you can check whether your installation recognizes your built-in NVIDIA GPU by running the following code in Python:

```
import torch
torch.cuda.is_available()
```

This returns

```
True
```

If the command returns `True`, you are all set. If the command returns `False`, your computer may not have a compatible GPU, or PyTorch does not recognize it. While GPUs are not required for the initial chapters in this book, which are focused on implementing LLMs for educational purposes, they can significantly speed up deep learning-related computations.

If you don't have access to a GPU, there are several cloud computing providers where users can run GPU computations against an hourly cost. A popular Jupyter notebook-like environment is Google Colab (<https://colab.research.google.com>), which provides time-limited access to GPUs as of this writing. Using the Runtime menu, it is possible to select a GPU, as shown in the screenshot in figure A.5.

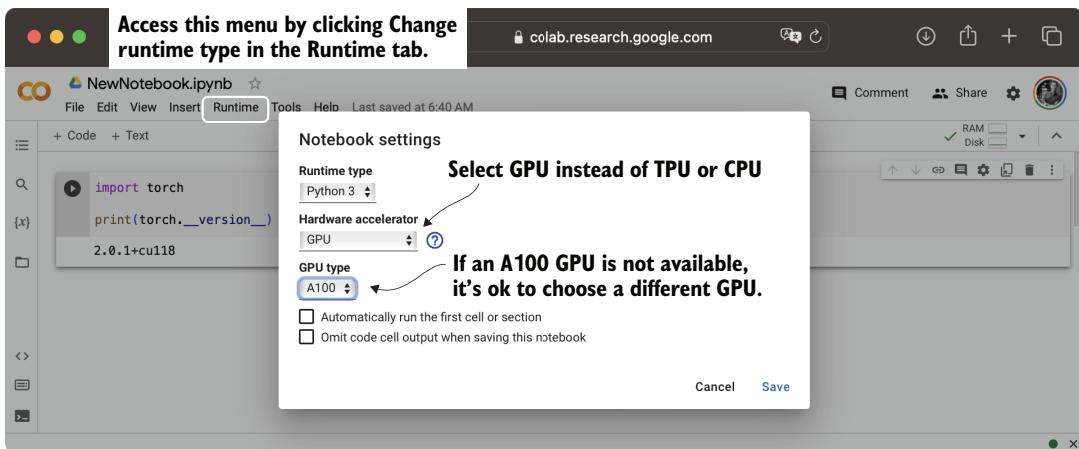


Figure A.5 Select a GPU device for Google Colab under the Runtime/Change Runtime Type menu.

PyTorch on Apple Silicon

If you have an Apple Mac with an Apple Silicon chip (like the M1, M2, M3, or newer models), you can use its capabilities to accelerate PyTorch code execution. To use your Apple Silicon chip for PyTorch, you first need to install PyTorch as you normally would. Then, to check whether your Mac supports PyTorch acceleration with its Apple Silicon chip, you can run a simple code snippet in Python:

```
print(torch.backends.mps.is_available())
```

If it returns `True`, it means that your Mac has an Apple Silicon chip that can be used to accelerate PyTorch code.

Exercise A.1

Install and set up PyTorch on your computer

Exercise A.2

Run the supplementary code at <https://mng.bz/o05v> that checks whether your environment is set up correctly.

A.2 Understanding tensors

Tensors represent a mathematical concept that generalizes vectors and matrices to potentially higher dimensions. In other words, tensors are mathematical objects that can be characterized by their order (or rank), which provides the number of dimensions. For example, a scalar (just a number) is a tensor of rank 0, a vector is a tensor of rank 1, and a matrix is a tensor of rank 2, as illustrated in figure A.6.

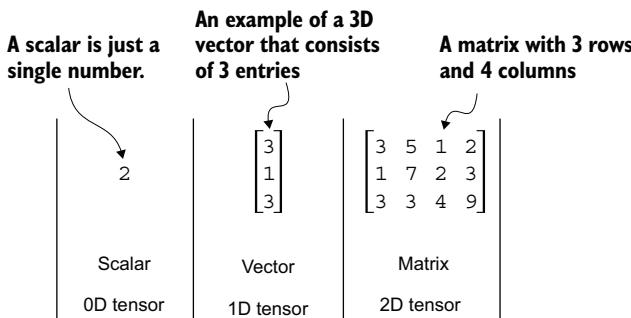


Figure A.6 Tensors with different ranks. Here 0D corresponds to rank 0, 1D to rank 1, and 2D to rank 2. A three-dimensional vector, which consists of three elements, is still a rank 1 tensor.

From a computational perspective, tensors serve as data containers. For instance, they hold multidimensional data, where each dimension represents a different feature. Tensor libraries like PyTorch can create, manipulate, and compute with these arrays efficiently. In this context, a tensor library functions as an array library.

PyTorch tensors are similar to NumPy arrays but have several additional features that are important for deep learning. For example, PyTorch adds an automatic differentiation engine, simplifying *computing gradients* (see section A.4). PyTorch tensors also support GPU computations to speed up deep neural network training (see section A.9).

PyTorch with a NumPy-like API

PyTorch adopts most of the NumPy array API and syntax for its tensor operations. If you are new to NumPy, you can get a brief overview of the most relevant concepts via my article “Scientific Computing in Python: Introduction to NumPy and Matplotlib” at <https://sebastianraschka.com/blog/2020/numpy-intro.html>.

A.2.1 Scalars, vectors, matrices, and tensors

As mentioned earlier, PyTorch tensors are data containers for array-like structures. A scalar is a zero-dimensional tensor (for instance, just a number), a vector is a one-dimensional tensor, and a matrix is a two-dimensional tensor. There is no specific term for higher-dimensional tensors, so we typically refer to a three-dimensional tensor as just a 3D tensor, and so forth. We can create objects of PyTorch’s `Tensor` class using the `torch.tensor` function as shown in the following listing.