Chpt3 Starting with Tensor

Augest 5, 2021

Covers:

1. Understanding tensors, the basic data structure in Pytorch
2. Indexing and operating on tensors
3. Interpolating with NumPy multidimensional arrays
4. Moving computations to the GPU for speed

3.1 The world as floating-point numbers

We need a way to encode real-world data of the kind we want to process into something digestible by a network (floating-point number), and then decode the ouput back to something we can understand and use for purpose.

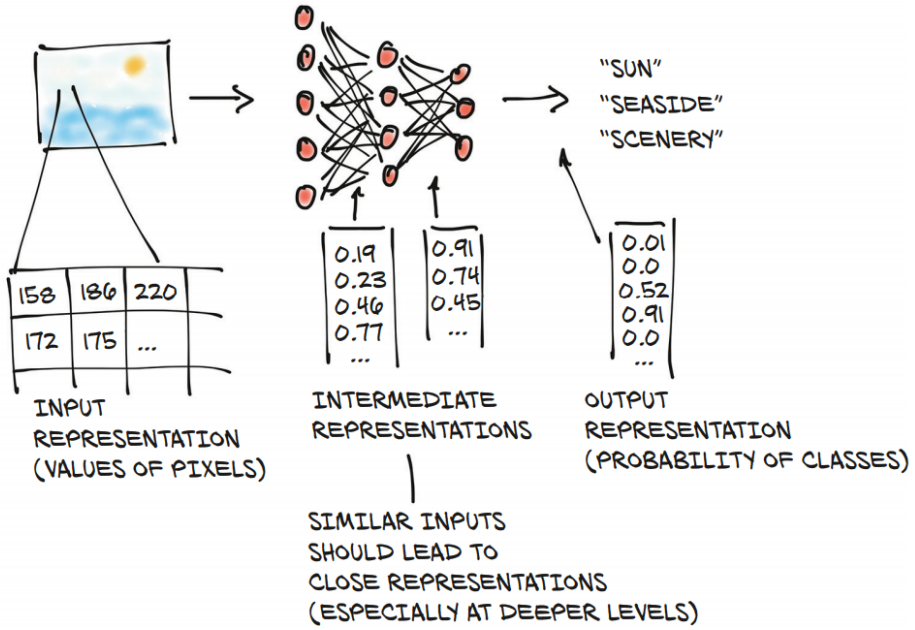

Figure 3.1 A deep neural network learns how to transform an input representation to an output representation. (Note: The numbers of neurons and outputs are not to scale.)

For image recognition, early representation can be things such as edge detection or certain texture like fur. Deeper representations can capture more complex structures like ears, noses, or eyes.

3.4 Named tensors

The dimensions of our tensors usually index something like pixel locations or color channels. This means when we want to index into a tensor, we need to remember the ordering of the dimensions

and write our indexing accordingly.

3.5 Tensor element types

So far, we have covered the basics of how tensors work, but we have not yet touched on what kinds of numeric types we can store in a *Tensor*. This section actually are more about the use of python.

Computations happening in neural network are typically executed with 32-bit floating-point precision. Creating a tensor with integers as arguments, such as using $torch.tensor([2, 2])$, will create a 64-bit integer tensor by default.

3.6 The tensor API

We mentioned the online docs earlier. They are exhausive and well-organized, with the tensor opeartions divided into groups:

1. $Creation - ops$: Functions for constructing tensor, like *ones* and $from\_numpy$
2. $Indexing, slicing, joining, mutating - ops$: Functions for changing the shape, stride, or content of a tensor, like *transpose*
3.