Pytorch Documentation

Chaitanya Kapoor

January 6, 2022

Contents

A list of important stuff in **PyTorch** that I do not want to keep using the official documentation for. I have made this using the book *"Deep Learning with PyTorch"*.

1 Tensor Manipulation

Tensors are indexed similar to multidimensional arrays. A Python list can also be passed to the constructor to create a tensor.

```
points = torch.tensor([4.0, 1.0, 5.0, 3.0, 2.0, 1.0])
```

The shape of the created tensor can also be accessed. Note that `shape` is **not** a callable function.

```
points.shape
```

Some common ways to index a tensor (analogous to lists):

```
points[1:]      # all rows after the first (implicitly all columns)
points[1:, :]   # all columns
points[1:, 0]   # first column
points[None]    # add a dimension of size 1 (like unsqueeze)
```

1.1 Tensor dimensions

3D tensors have shapes in the following order:

```
img_t = torch.randn(3, 5, 5) # shape [channels, rows, columns]
```

A 4D image tensor has the following shape:

```
batch_t = torch.randn(2, 3, 5, 5) # shape [batch, channels, rows, columns]
```

RGB dimensions are always counted third from the end, **-3**.

1.2 In-place operations

Methods which are specific to `Tensor` objects are followed by an *underscore*. This is used to indicate that the method operates *in place* by modifying the input instead of creating a new tensor.

```
a = torch.ones(3,2)
a.zero_()
```

This code block mutates `a` and returns a null tensor of shape $(3, 2)$.

1.3 Move to GPUs

A tensor can be created on the GPU by providing the following argument to the constructor:

```
points_gpu = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]], device='cuda')
```

We can also copy a tensor created on the CPU to the GPU with the help of the `to` method.

```
points_gpu = points.to(device='cuda')
```

Basic tensor operations can also be transferred to the GPU rather than performing them on the CPU.

```
points_gpu = 2 * points.to(device='cuda')
```

1.4 Serializing tensors

To avoid retraining models from scratch, we can save weight tensors with the help of the `pickle` format. If we have an output file called `ourpoints.t`, then:

```
torch.save(points, '../ourpoints.t')
```

To load the weights, we call the `load` method.

```
points = torch.load('..ourpoints.t')
```

2 Images

2.1 Loading images

This can be done with the `imageio` library. Suppose that the path to the image file is `'../xyz.jpg'`:

```
import imageio
img_arr = imageio.imread('../xyz.jpg')
```

The image is loaded as a `NumPy` object. The layout of the dimensions is $H \times W \times C$ corresponding to **height**, **width** and **channel** respectively.

2.2 Changing layout

The layout of the image tensor can be altered using the `permute` method (as expected by `Pytorch`).

```
torch_img = torch.from_numpy(img_arr) # convert to torch tensor
out = torch_img.permute(2, 0, 1) # (C X H X W)
```

An extra dimension can be added for processing batches of images by specifying the dimensions of the tensor as $N \times C \times H \times W$ where N is the batch size of the images.