for it. We could do the same for the **validation** data, using the **split** we performed at the beginning of this book, or we could use `random_split()` instead.

Random Split

PyTorch's `random_split()` method is an easy and familiar way of performing a **training-validation split**.

So far, we've been using `x_train_tensor` and `y_train_tensor`, built out of the original split in *Numpy*, to build the **training dataset**. Now, we're going to be using the **full data** from *Numpy* (`x` and `y`) to build a PyTorch `Dataset` **first** and only then **split** the data using `random_split()`.

Although there was a (funny) reasoning behind my choice of 42 as a random seed, I'll be using other numbers as seeds, mostly **odd numbers**, just because I like them better :-)

Since v1.13, PyTorch's `random_split()` method takes fractions as arguments for the split (similarly to Scikit-Learn's `train_test_split()`). In the example that follows, it wouldn't be necessary to manually compute `n_train` and `n_val` anymore. We could simply use the `ratio` directly (the fractions need to add up to one):

```
ratio = .8
eps = 1e-16
train_data, val_data = random_split(
    dataset,
    [ratio, 1-ratio+eps]
)
```

You are probably wondering what that `eps` is doing there, right? As it turns out, `random_split()` rounds down the number of elements in each subset which may lead to somewhat unexpected results (e.g. 19 data points in the validation set) because of precision issues (`1-ratio` equals `0.19999999999999996`). Adding `eps` to the remainder prevents this from happening (as long as it's added **at the end** of the expression).

Run - Model Training V4

```
%run -i model_training/v4.py
```

After updating all parts, in sequence, our current state of development is:

- **Data Preparation V2**
- **Model Configuration V2**
- **Model Training V4**

Let's inspect the model's state:

```
# Checks model's parameters
print(model.state_dict())
```

Output

```
OrderedDict([('0.weight', tensor([[1.9419]], device='cuda:0')),
('0.bias', tensor([1.0244], device='cuda:0'))])
```

As of version 1.9, PyTorch offers a new context manager: `torch.inference_mode()`. It also disables gradient computation but it goes one step further and disables PyTorch's internal view tracking as well thus delivering better performance. In the examples used in this book, however, the difference is negligible.