# Chapter 1

# An Introduction to PyTorch

## 1.1 A Fun Example

Efficient machine learning processes data in batches, and our model will expect a batch of data.

We use PyTorch's `unsqueeze()` function to add a dimension to our tensor and create a batch of size 1.

The use of `model.to(device)` and `batch.to(device)` sends our model and input data to the GPU if available, and executing `model(batch.to(device))` runs our classifier.

# Chapter 2

# Tensors

## 2.1 Creating Tensors

Use torch.arange() when the step size is known. Use torch.linspace() when the number of elements is known. You can use torch.tensor() to create tensors from array-like structures such as lists, NumPy arrays, tuples, and sets. To convert existing tensors to NumPy arrays and lists, use the torch.numpy() and torch.tolist() functions, respectively.

### 2.1.1 Data Types

To reduce space complexity, you may sometimes want to reuse memory and overwrite tensor values using in-place operations. To perform in-place operations, append the underscore (_) postfix to the function name. For example, the function y.add_(x) adds x to y, but the results will be stored in y.

### 2.1.2 Creating Tensors from Random Samples

Table: Random sampling functions

### 2.1.3 Creating Tensors Like Other Tensors

You may want to create and initialize a tensor that has similar properties to another tensor, including the dtype, device, and layout properties to facilitate calculations. Many of the tensor creation operations have a similarity function that allows you to easily do this. The similarity functions will have the postfix _like. For example, torch.empty_like(tensor_a) will create an empty tensor with the dtype, device, and layout properties of tensor_a. Some examples of similarity functions include empty_like(), zeros_like(), ones_like(), full_like(), rand_like(), randn_like(), and rand_int_like().

## 2.2 Tensor Operations

Table 2.1: Tensor creation functions

| Function | Description |
|---|---|
| torch.tensor(data, dtype=None, device=None, requires_grad=False, pin_memory=False) | Creates a tensor from an existing data structure |
| torch.empty(*size, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a tensor from uninitialized elements based on the random state of values in memory |
| torch.zeros(*size, out=None, dtype=None, layout=torch.strided, evice=None, requires_grad=False) | Creates a tensor with all elements initialized to 0.0 |
| torch.ones(*size, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a tensor with all elements initialized to 1.0 |
| torch.arange(start=0, end, step=1, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a 1D tensor of values over a range with a common step value |
| torch.linspace(start, end, steps=100, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a 1D tensor of linearly spaced points between the start and end |
| torch.logspace(start, end, steps=100, base=10.0, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a 1D tensor of logarithmically spaced points between the start and end |
| torch.eye(n, m=None, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a 2D tensor with ones on the diagonal and zeros everywhere else |
| torch.full(size, fill_value, out=None, dtype=None, layout=torch.strided, device=None, requires_grad=False) | Creates a tensor filled with fill_value |
| torch.load(f) | Loads a tensor from a serialized pickle file |
| torch.save(f) | Saves a tensor to a serialized pickle file |