Resource :- https://www.learnpytorch.io/

In [1]:
```
import torch
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
print(torch.__version__)
```

```
1.12.1
```

Introduction to Tensors

Creating tensors

PyTorch tensors are created using torch.Tensor() = https://www.learnpytorch.io/00_pytorch_fundamentals/

In [3]:
```
# Scalar
scalar = torch.tensor(7)
scalar
```

Out[3]:
```
tensor(7)
```

In [4]:
```
scalar.ndim
```

Out[4]:
```
0
```

In [5]:
```
# Get tensor back as Python int (only works with one-element tensors)
scalar.item()
```

Out[5]:
```
7
```

In [6]:
```
# vector
vector = torch.tensor([4,5])
vector
```

Out[6]:
```
tensor([4, 5])
```

In [7]:
```
# Check the number of dimensions of vector
vector.ndim
```

Out[7]:
```
1
```

In [8]:
```
# Check shape of vector
vector.shape
```

Out[8]:
```
torch.Size([2])
```

The above returns torch.Size([2]) which means our vector has a shape of [2]. This is because of the two elements we placed inside the square brackets ([4, 5]).

In [9]:
```
# Matrix
MATRIX = torch.tensor([[3,4],
                       [5,6]])
MATRIX
```

Out[9]:
```
tensor([[3, 4],
        [5, 6]])
```

In [10]:
```
# Check number of dimensions
MATRIX.ndim
```

Out[10]:
```
2
```

In [11]:
```
MATRIX.shape
```

Out[11]:
```
torch.Size([2, 2])
```

In [12]:
```
MATRIX[0]
```

Out[12]:
```
tensor([3, 4])
```

We get the output torch.Size([2, 2]) because MATRIX is two elements deep and two elements wide.

In [13]:
```
# Tensor
TENSOR = torch.tensor([[[1, 2, 3],
                        [3, 6, 9],
                        [2, 4, 5]]])
TENSOR
```

Out[13]:
```
tensor([[[1, 2, 3],
         [3, 6, 9],
         [2, 4, 5]]])
```

In [14]:
```
# Check number of dimensions for TENSOR
TENSOR.ndim
```

Out[14]:
```
3
```

In [15]:
```
# Check shape of TENSOR
TENSOR.shape   #[colour_channels, height, width]
```

Out[15]:
```
torch.Size([1, 3, 3])
```

In [16]:
```
TENSOR[0]
```

Out[16]:
```
tensor([[1, 2, 3],
        [3, 6, 9],
        [2, 4, 5]])
```

Random tensors

Why random tensors?

Random tensors are important because the way many neural networks learn is that they start with ramdom nembers and then adjust those random numbers to better represent the data.

`Start with random numbers -> look at data -> update random numbers -> look at data -> update random numbers`

Torch random tensor - https://pytorch.org/docs/stable/generated/torch.rand.html

In [17]:
```
# Create a random tensor of size (4,5)
random_tensor = torch.rand(4,5)
```

In [18]:
```
random_tensor
```

Out[18]:
```
tensor([[0.4159, 0.2099, 0.5318, 0.4120, 0.1086],
        [0.5739, 0.8637, 0.0190, 0.4905, 0.7357],
        [0.2774, 0.1716, 0.4907, 0.3476, 0.0808],
        [0.0976, 0.4561, 0.8523, 0.4610, 0.3667]])
```

In [19]:
```
random_tensor.ndim
```

Out[19]:
```
2
```

In [20]:
```
random_tensor = torch.rand(1,4,5)
random_tensor
```

Out[20]:
```
tensor([[[0.4020, 0.6127, 0.9032, 0.3627, 0.9306],
         [0.5488, 0.2449, 0.6947, 0.0885, 0.9659],
         [0.9436, 0.9642, 0.9046, 0.2522, 0.1258],
         [0.2062, 0.7892, 0.2590, 0.0832, 0.9909]]])
```

In [21]:
```
random_tensor.ndim
```

Out[21]:
```
3
```

In [22]:
```
# Create a random tensor with similar shape to an image tensor
random_image_size_tensor = torch.rand(size=(3,224,224)) # Colour channel,height,width
random_image_size_tensor.shape, random_image_size_tensor.ndim
```

Out[22]:
```
(torch.Size([3, 224, 224]), 3)
```

Zeros and ones

In [23]:
```
# Creating a tensor of all zeros
zeros = torch.zeros(size=(4,5))
zeros
```

Out[23]:
```
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]])
```

In [24]:
```
zeros* random_tensor
```

```
zeros, random_tensor
```

Out[24]:
```
tensor([[[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]]])
```

In [25]:
```
# Create a tensor of all ones
ones = torch.ones(size=(4,5))
ones
```

Out[25]:
```
tensor([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])
```

In [26]:
```
ones.dtype
```

Out[26]:
```
torch.float32
```

In [27]:
```
random_tensor.dtype
```

Out[27]:
```
torch.float32
```

In [28]:
```
## Creating a range of tensors and tensors-like
```

In [29]:
```
# Use torch.range()
one_to_ten = torch.arange(start=0,end=11,step=1) #torch.arange(1,11)
one_to_ten
```

Out[29]:
```
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [30]:
```
# Creating tensors like (Returns a tensor filled with the scalar value 0, with the same size as in one_to_ten)
ten_zeros = torch.zeros_like(input=one_to_ten)
ten_zeros
```

Out[30]:
```
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Tensor datatypes

Resource :- https://pytorch.org/docs/stable/tensors.html

Note :- Tensor datatypes is one of the 3 big errors you'll run into with PyTorch and deep learning:

1. Tensors not right datatype
2. Tensors not right shape
3. Tensors not on the right device

In [31]:
```
# Float 32 tensor
float_32_tensor = torch.tensor([1.0,3.4,5.6], dtype=None)
float_32_tensor
```

Out[31]:
```
tensor([1.0000, 3.4000, 5.6000])
```

In [32]:
```
float_32_tensor.dtype # If dtype is given none then also the default type is float32
```

Out[32]:
```
torch.float32
```

In [33]:
```
# Float 16 tensor
float_16_tensor = torch.tensor([1.0,3.4,5.6], dtype=torch.float16)
float_16_tensor.dtype
```

Out[33]:
```
torch.float16
```

Single precision is 32-bit and half precision is 16-bit, but if we require more precise value then we have to go with 32-bit otherwise in term of memory space it is better to go with 16bit

In [34]:
```
float_32_tensor = torch.tensor([3.0,6.0,9.0],
                               dtype=None, #What datatype is the tensor (e.g. float32 or float16)
                               device=None, #What device is your tensor on like GPU or CPU
                               requires_grad = False) # Whether or not to track gradients with this operations
float_32_tensor
```

Out[34]:
```
tensor([3., 6., 9.])
```

In [35]:
```
float_32_tensor.dtype
```

Out[35]:
```
torch.float32
```

In [36]:
```
float_16_tensor = float_32_tensor.type(torch.float16)
```

```
In [ ]: float_16_tensor
```

```
Out[36]: tensor([3., 6., 9.], dtype=torch.float16)
```

```
In [37]: float_16_tensor * float_32_tensor
```

```
Out[37]: tensor([ 9., 36., 81.])
```

```
In [38]: int_32_tensor = torch.tensor([3,4,5], dtype=torch.int32)
         int_32_tensor
```

```
Out[38]: tensor([3, 4, 5], dtype=torch.int32)
```

```
In [39]: float_32_tensor * int_32_tensor # It may produce error when we perform multiplication of two different datatype
```

```
Out[39]: tensor([ 9., 24., 45.])
```

Getting information from tensors

1. To get datatype from a tensor, we can use `tensor.dtype`
2. To get shape frem a tensor, we can use `tensor.shape`
3. To get device from a tensor, we can use `tensor.device`

```
In [40]: # Create a tensor
         some_tensor = torch.rand(3,4)
         some_tensor
```

```
Out[40]: tensor([[0.5260, 0.6779, 0.8463, 0.9223],
                 [0.8965, 0.3267, 0.3122, 0.6596],
                 [0.9975, 0.0284, 0.8218, 0.7520]])
```

```
In [41]: # Find out details about some tensor
         print(some_tensor)
         print(f"Datatype of tensor: {some_tensor.dtype}")
         print(f"Shape of tensor: {some_tensor.shape}")  # Attribute
         print(f"Shape of tensor: {some_tensor.size()}") # Function
         print(f"Device tensor is on: {some_tensor.device}") # By default it is in CPU
```

```
tensor([[0.5260, 0.6779, 0.8463, 0.9223],
        [0.8965, 0.3267, 0.3122, 0.6596],
        [0.9975, 0.0284, 0.8218, 0.7520]])
Datatype of tensor: torch.float32
Shape of tensor: torch.Size([3, 4])
Shape of tensor: torch.Size([3, 4])
Device tensor is on: cpu
```

Manipulating Tensors (Tensor operations)

Tensor operations include:

1. Addition
2. Subtraction
3. Multiplication (element-wise)
4. Division
5. Matrix multiplication

```
In [42]: # Create a tensor and add 10 to it
         tensor = torch.tensor([5,6,7])
         tensor + 10
```

```
Out[42]: tensor([15, 16, 17])
```

```
In [43]: # Multiply tensor by 10
         tensor * 10
```

```
Out[43]: tensor([50, 60, 70])
```

```
In [44]: tensor
```

```
Out[44]: tensor([5, 6, 7])
```

```
In [45]: # Subtract 10
         tensor - 10
```

```
Out[45]: tensor([-5, -4, -3])
```

```
In [46]: # Try out PyTorch in-built functions
```

```
torch.mul(tensor, 10)
```

Out[46]:
```
tensor([50, 60, 70])
```

In [47]:
```
torch.add(tensor,10)
```

Out[47]:
```
tensor([15, 16, 17])
```

Matrix multiplication

Two main ways of performing multiplication in neural networks and deep learning:

1. Element-wise multiplication
2. Matrix multiplication(dot product)

In [48]:
```
# Element wise multiplication
print(tensor, "*", tensor)
print(f"Equals: {tensor * tensor}")
```

```
tensor([5, 6, 7]) * tensor([5, 6, 7])
Equals: tensor([25, 36, 49])
```

In [49]:
```
# Matrix multiplication
torch.matmul(tensor,tensor)
```

Out[49]:
```
tensor(110)
```

In [50]:
```
tensor
```

Out[50]:
```
tensor([5, 6, 7])
```

In [51]:
```
%%time
value = 0
for i in range(len(tensor)): # Matrix multiplication using loop
    value += tensor[i] * tensor[i]
print(value)
```

```
tensor(110)
CPU times: total: 0 ns
Wall time: 0 ns
```

In [52]:
```
%%time
torch.matmul(tensor, tensor) # faster
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

Out[52]:
```
tensor(110)
```

In [ ]:
```

```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js