Conventions & Notations

Suleyman Demirel University

CSS634: Deep Learning

PhD Abay Nussipbekov

Scalar

- Rank-0 tensor

  $$x \in \mathbb{R}$$

- Example: $x = 1$

Vector

- Rank-1 tensor
- $\mathbf{x} \in \mathbb{R}^n$ ← Sometimes D or m

  $\mathbf{x} \in \mathbb{R}^{n\times 1}$ ← You will see column representation often
- Example: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
- $\mathbf{x}^{\mathrm{T}} = [x_1 \quad x_2 \quad \cdots \quad x_n]$, where $\mathbf{x}^{\mathrm{T}} \in \mathbb{R}^{1\times n}$

Matrix

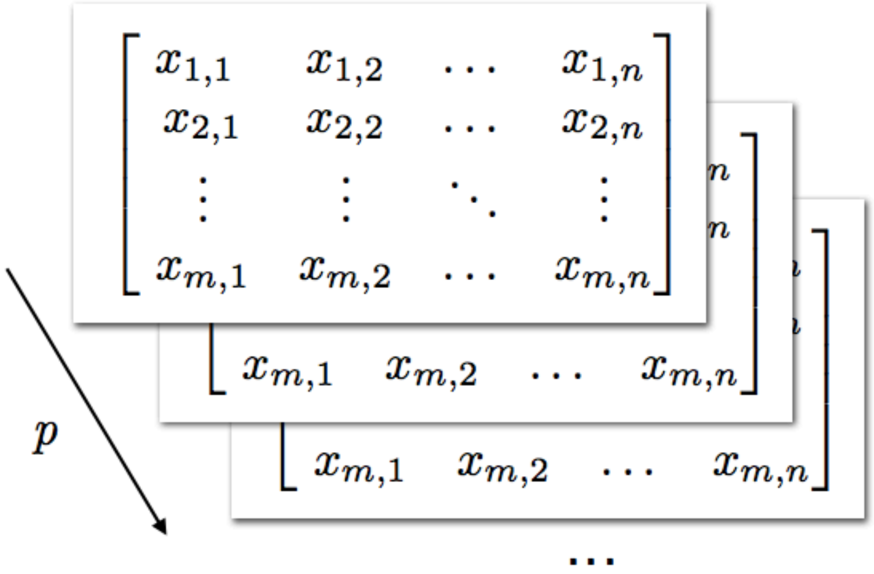
- Rank-2 tensor
- $\mathbf{X} \in \mathbb{R}^{m \times n}$ Sometimes D or m

  $x \in \mathbb{R}^{n \times 1}$ You will see column representation often
- Example: $\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix}$
- In many literatures n stands for number of examples so the shape of matrix (design matrix) is $X \in \mathbb{R}^{n \times m}$
- In x_i the subscript i denotes i^{th} feature

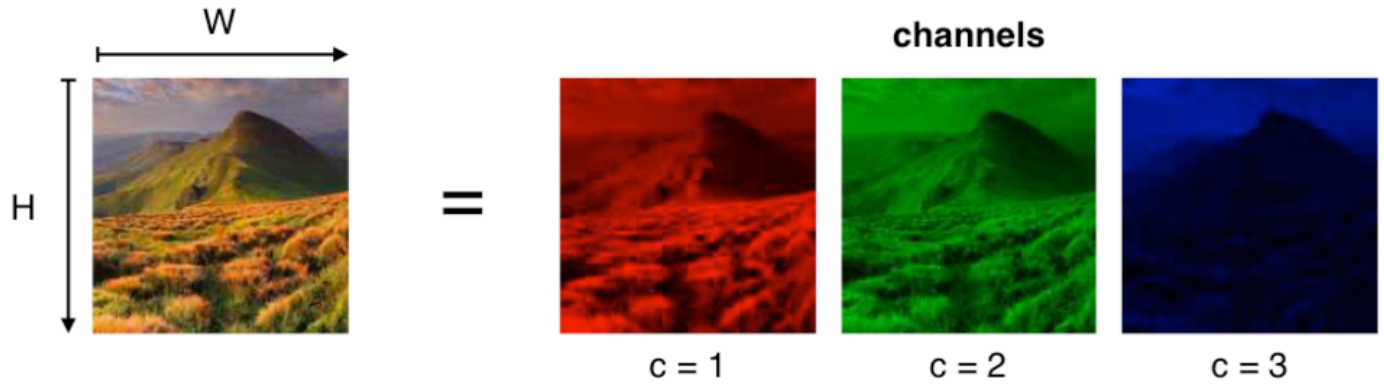
Tensor

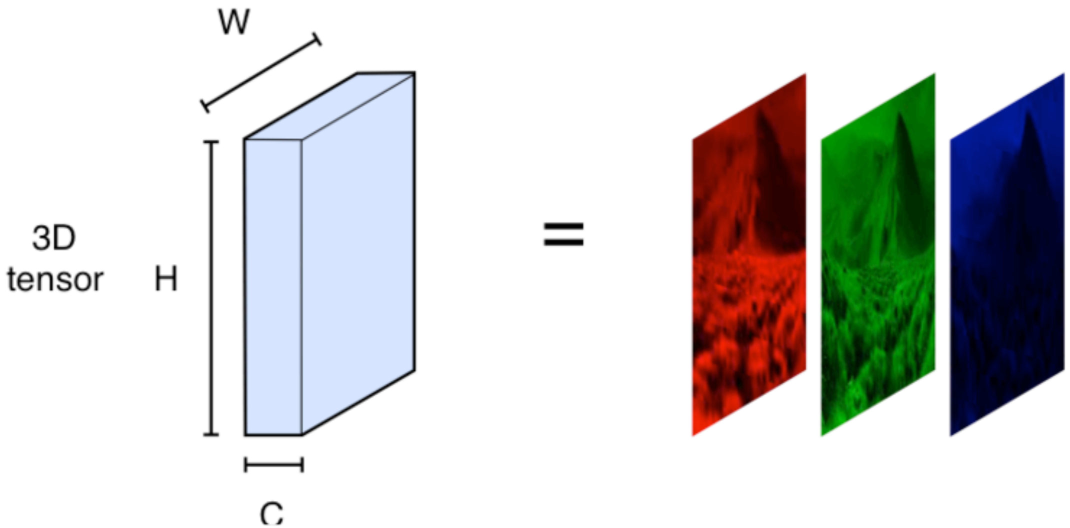
- A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array.
- *In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a tensor*
- Rank-3 tensor

  $\mathbf{X} \in \mathbb{R}^{m \times n \times p}$



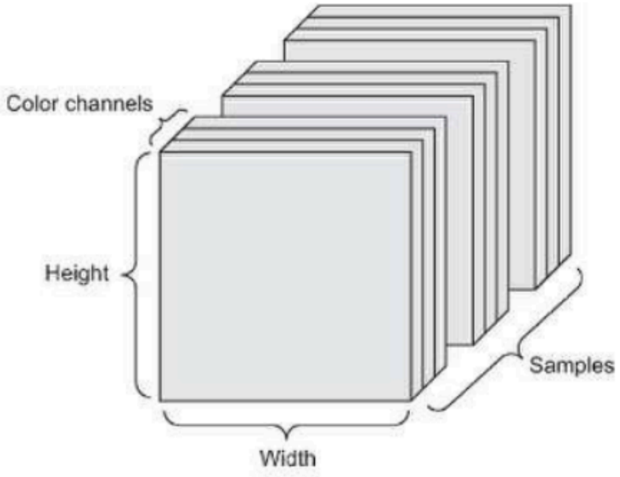
3D Tensor





4D Tensor

Example : A batch of 128 color images of size 256 * 256



Shape : (samples, height, width, channels)
= (128, 256, 256, 3)

Multidimensional Arrays as Tensors

In [1]:

```
import numpy as np
import torch
```

In [2]:

```
a = np.array([1., 2., 3.])
b = torch.tensor([1., 2., 3.])
```

In [3]:

```
print(a.dtype)
print(b.dtype)
```

```
float64
torch.float32
```

In [4]:

```
print(a.shape)
print(b.size())
```

```
(3,)
torch.Size([3])
```

NumPy vs. PyTorch

```
In [3]: import numpy as np
        import torch

In [4]: a = np.array([1., 2., 3.])
        b = torch.tensor([1., 2., 3.])

In [5]: print(a.dot(a))

        14.0

In [6]: print(b.matmul(b))

        tensor(14.)

In [7]: b.numpy()

Out[7]: array([1., 2., 3.], dtype=float32)

In [8]: torch.tensor(a)

Out[8]: tensor([1., 2., 3.], dtype=torch.float64)
```

“dot” vs “matmul”

We can convert, but pay attention to default types

Data Types to Memorize

NumPy data type	Tensor data type	
`numpy.uint8`	`torch.ByteTensor`	
`numpy.int16`	`torch.ShortTensor`	
`numpy.int32`	`torch.IntTensor`	
`numpy.int`	`torch.LongTensor`	
`numpy.int64`	`torch.LongTensor`	default int in NumPy & PyTorch
`numpy.float16`	`torch.HalfTensor`	
`numpy.float32`	`torch.FloatTensor`	default float in PyTorch
`numpy.float`	`torch.DoubleTensor`	
`numpy.float64`	`torch.DoubleTensor`	default float in NumPy

- E.g., `int32` stands for 32 bit integer
- 32 bit floats are less precise than 64 floats, but for neural nets, it doesn't matter much
- For regular GPUs, we usually want 32 bit floats (vs 64 bit floats) for fast performance

Why Not Just NumPy?

- PyTorch has GPU support:
  A. we can load the dataset and model parameters into GPU memory
  B. on the GPU we then have better parallelism for computing (many) matrix multiplications
- Also, PyTorch has automatic differentiation (more later)
- Moreover, PyTorch implements many DL convenience functions (more later)

Loading Data onto the GPU

```
[ ]  import numpy as np
     import torch
```

```
[ ]  # check if GPU is available
     print(torch.cuda.is_available())
```

```
True
```

```
[ ]  # choose the available device
     device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
[ ]  a = torch.tensor([1., 2., 3.], dtype=torch.float)
     a.to(torch.device('cuda:0'))
```

```
tensor([1., 2., 3.], device='cuda:0')
```

Vectorization

- Suppose we need to calculate a linear transformation: $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{w} + b$

  where X: $\begin{bmatrix} x_1^{(1)} & \cdots & x_m^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \cdots & x_m^{(n)} \end{bmatrix}$ and w: $\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$

- The resultant matrix Z is then: $\begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(n)} \end{bmatrix}$

- Bias b is scalar which will be automatically "broadcasted": $b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$

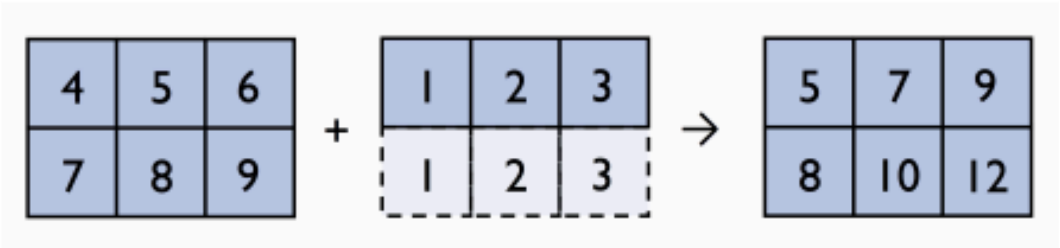
Broadcasting

```
In [4]: torch.tensor([1, 2, 3]) + 1
Out[4]: tensor([2, 3, 4])
```


```
In [5]: t = torch.tensor([[4, 5, 6], [7, 8, 9]])

In [6]: t
Out[6]:
tensor([[4, 5, 6],
        [7, 8, 9]])

In [7]: t + torch.tensor([1, 2, 3])
Out[7]:
tensor([[ 5,  7,  9],
        [ 8, 10, 12]])
```



Implicit dimensions get added,
elements are implicitly duplicated

Resources Used

- STAT 479: Deep Learning by Sebastian Raschka
- Pytorch.org
- Machinelearningmastery.com by Jason Brownlee
- Deeplearningbook by Ian Godfellow