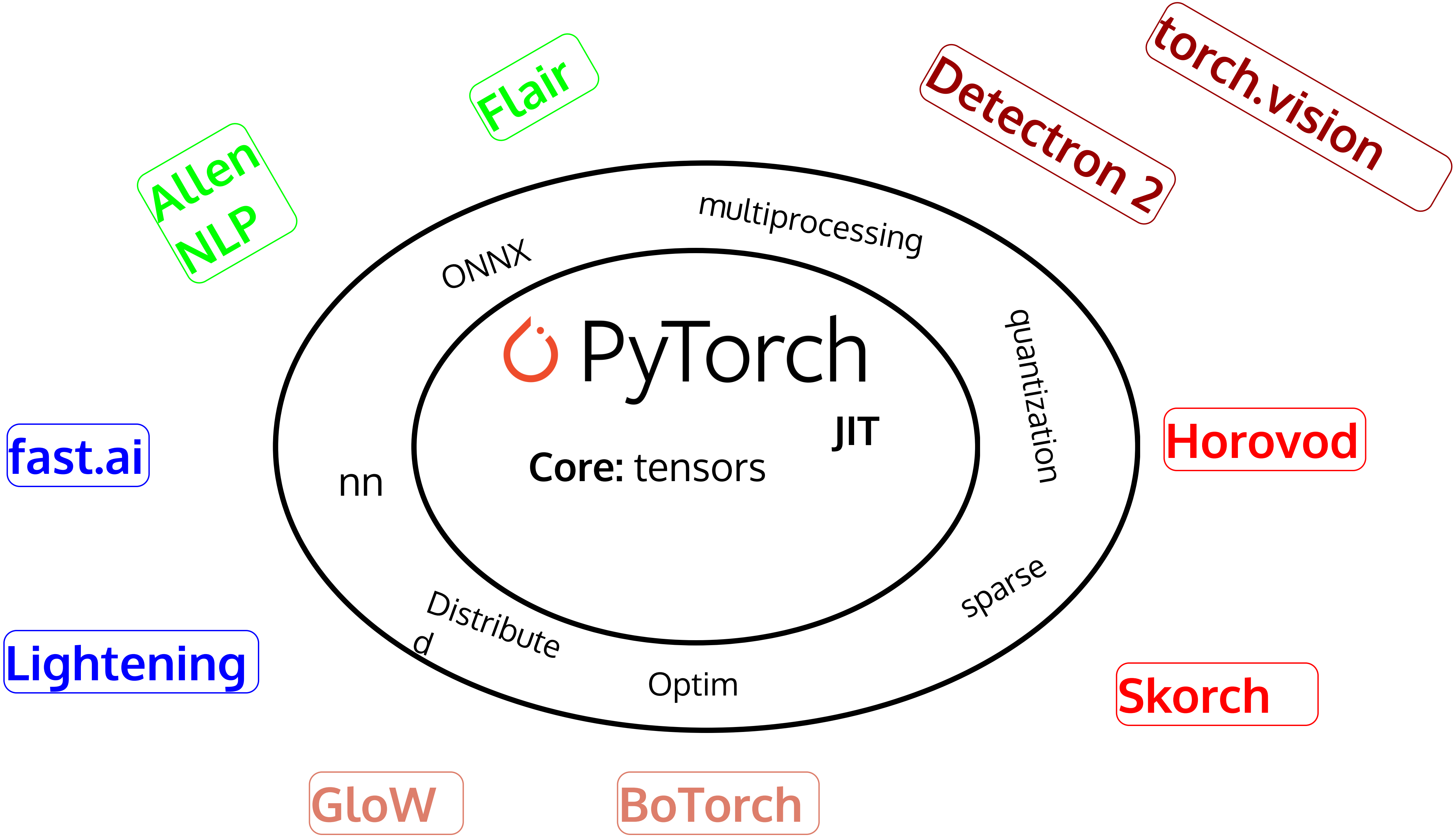
Developing Deep Learning Models using

PyTorch

Arun Prakash A

DL Course Instructor

BS, IIT Madras, POD


PyTorch
Core: tensors
JIT
nn
ONNX
multiprocessing
quantization
sparse
Optim
Distribute
d
Allen
NLP
Flair
Detectron 2
torch.vision
fast.ai
Lightening
Horovod
Skorch
GloW
BoTorch

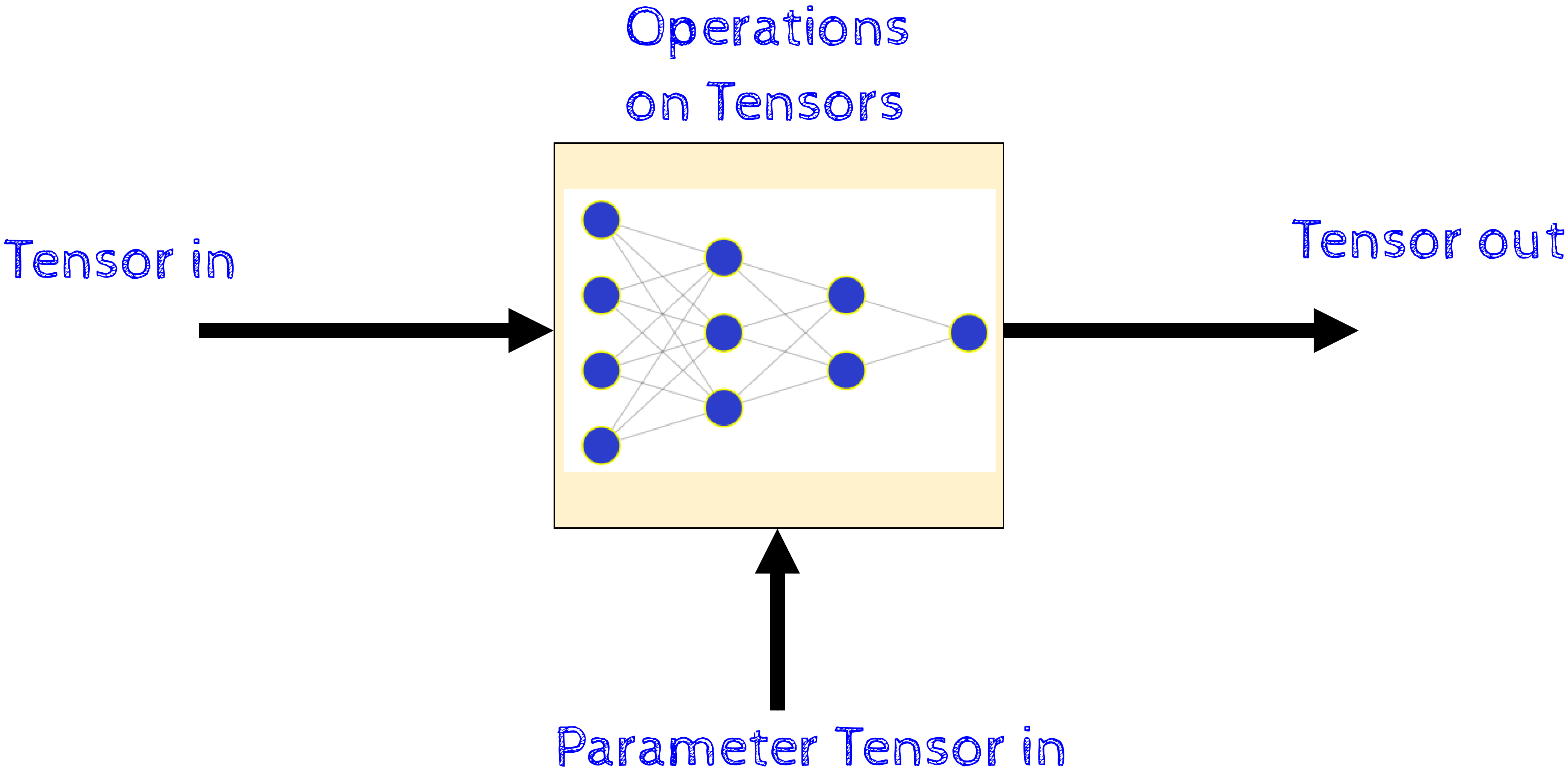
The objective is not only Build-Train-Test but also to Debug Deep Learning models

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):

    def __init__(self, num_hidden):
        super(Model, self).__init__()
        self.layer1 = nn.Linear(28 * 28, 100)
        self.layer2 = nn.Linear(100, 50)
        self.layer3 = nn.Linear(50, 20)
        self.layer4 = nn.Linear(20, 1)
        self.num_hidden = num_hidden
    def forward(self, img):
        flattened = img.view(-1, 28 * 28)
        activation1 = F.relu(self.layer1(flattened))
        activation2 = F.relu(self.layer2(activation1))
        activation3 = F.relu(self.layer3(activation2))
        output = self.layer4(activation3)
        return output
```

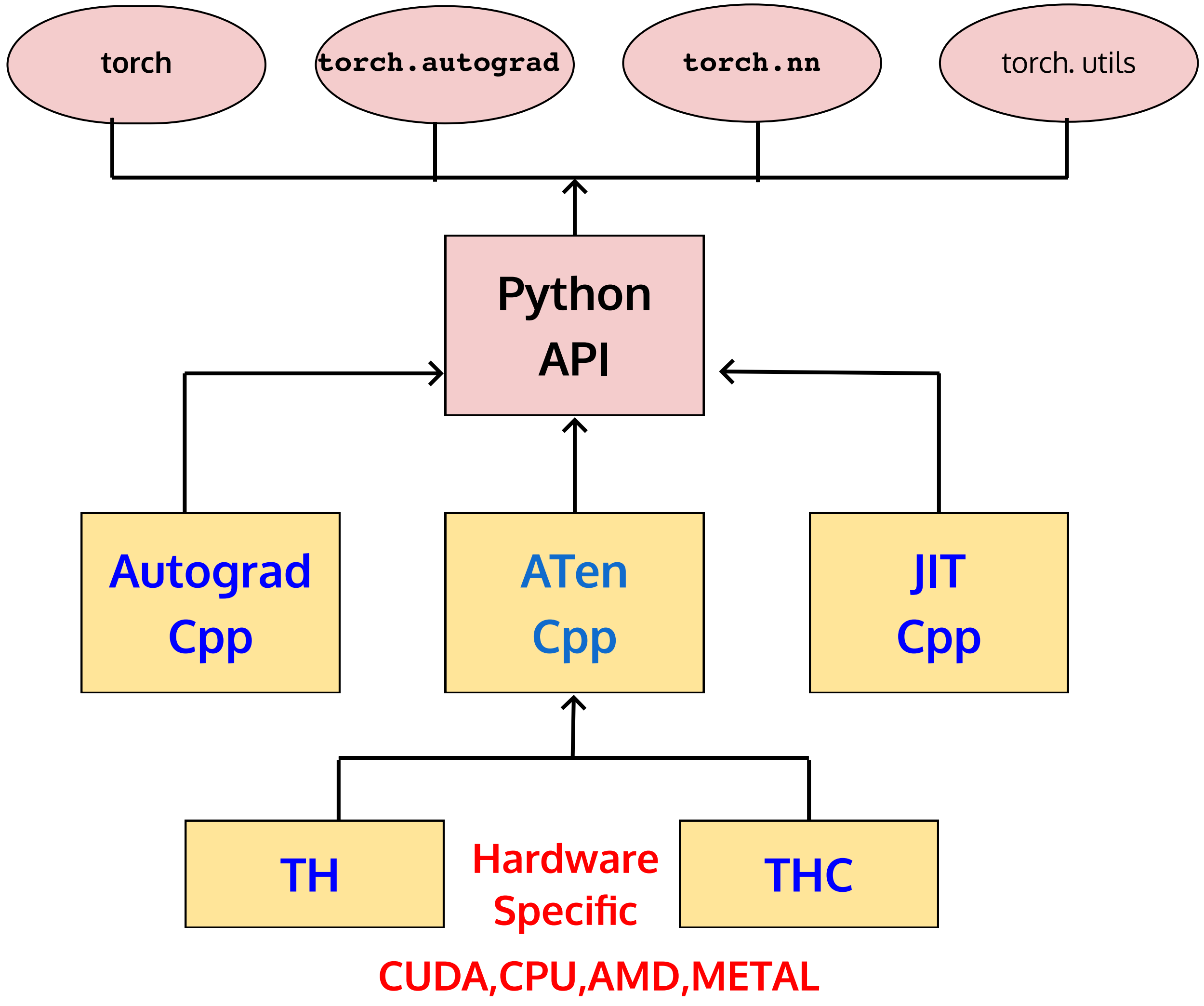
Debugging requires a deeper understanding of things happening under the hood!

For the next two sessions, you most likely feel you are not doing deep learning 😃

The Software Architecture of PyTorch


Operations
on Tensors
Tensor in
Tensor out
Parameter Tensor in

"A Tensor": ATen
"Caffe2 10": C10



Tensor computation (like NumPy) with strong GPU acceleration

Deep neural networks built on a tape-based autograd system

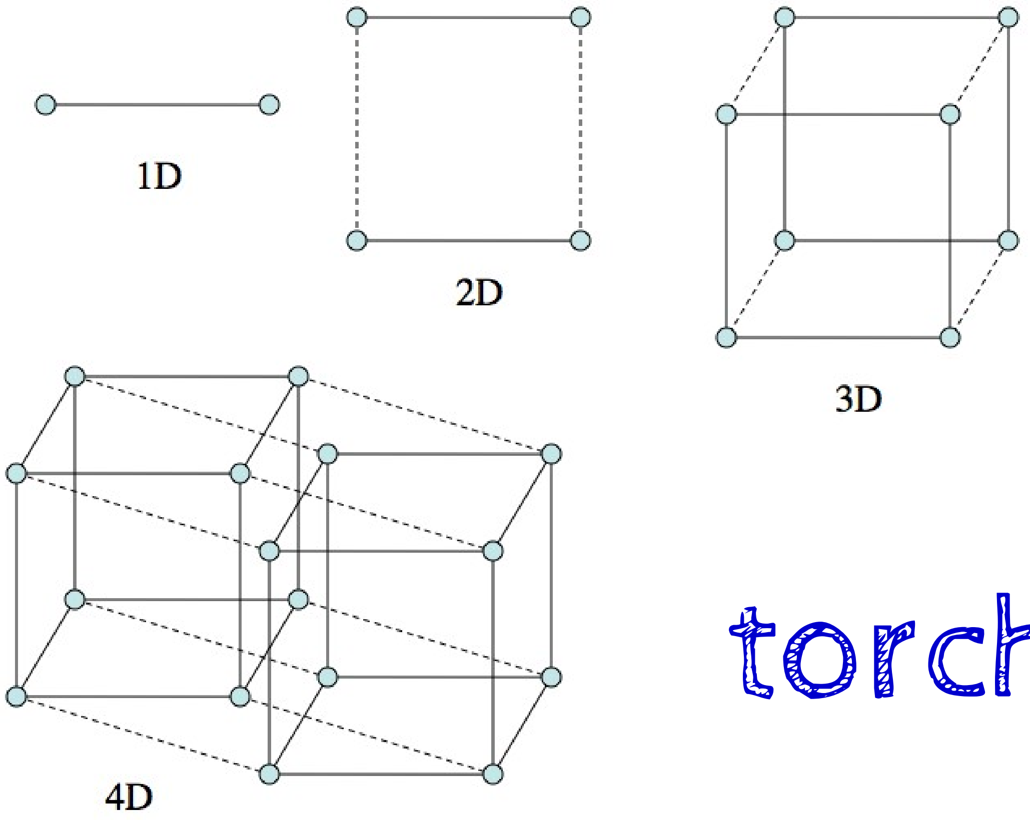
Support efficient industry production at massive scale

Support exporting models to Python-less environment

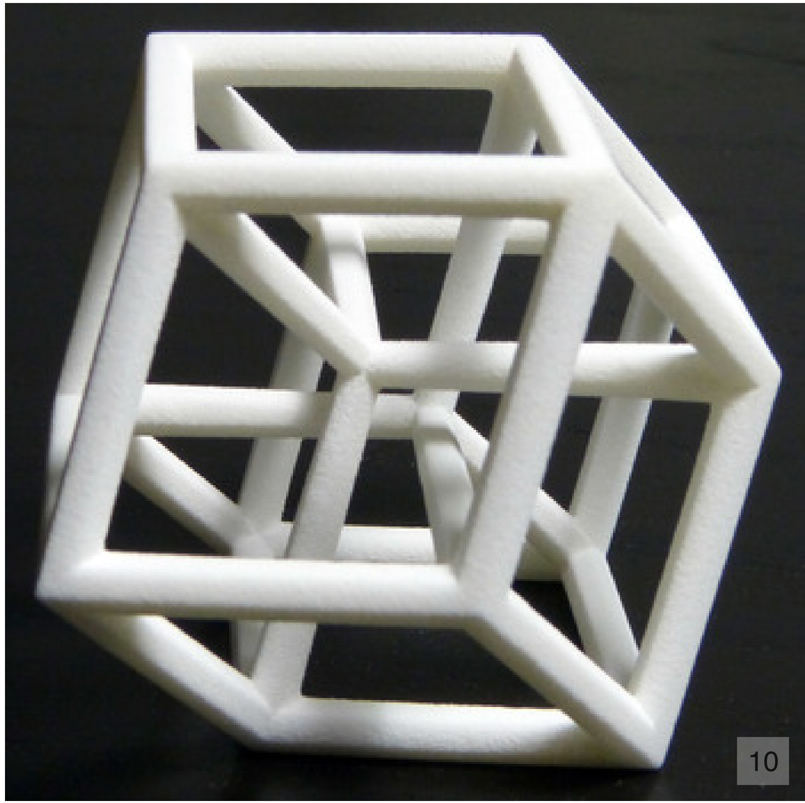
Support for platforms of Caffe2 (iOS, Android, Raspbian, Tegra, etc) and will continue to expand various platforms support

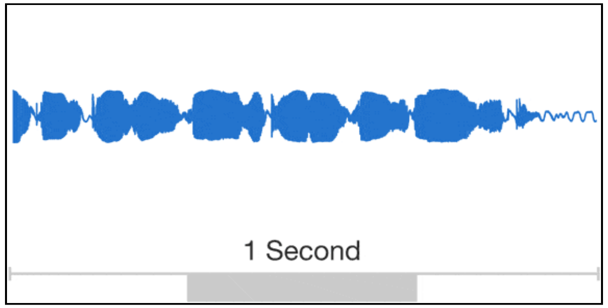
Component	Description
torch	A Tensor library like NumPy, with strong GPU support
torch.autograd	A tape-based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.jit	A compilation stack (TorchScript) to create serializable and optimizable models from PyTorch code
torch.nn	A neural networks library deeply integrated with autograd designed for maximum flexibility
torch.multiprocessing	Python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and Hogwild training
torch.utils	DataLoader and other utility functions for convenience

https://www.youtube.com/embed/DSgJ1sejWtw?enablejsapi=1

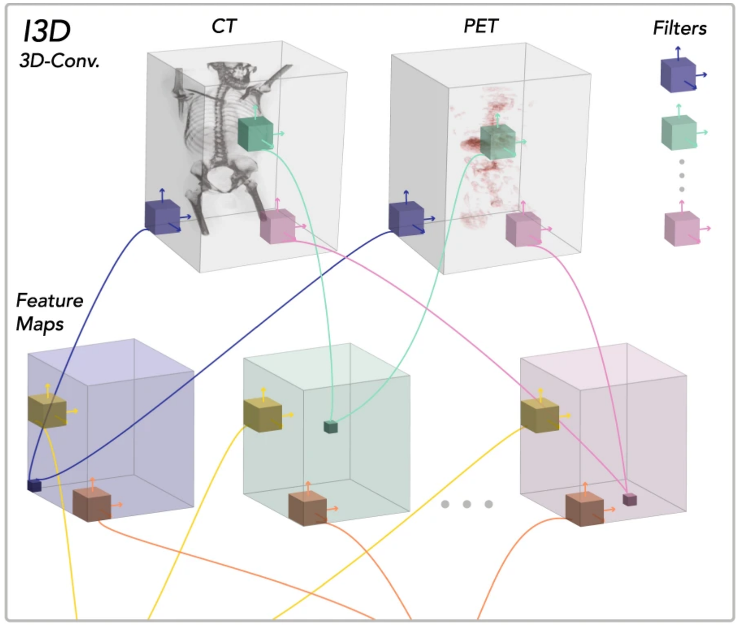


torch.tensor()



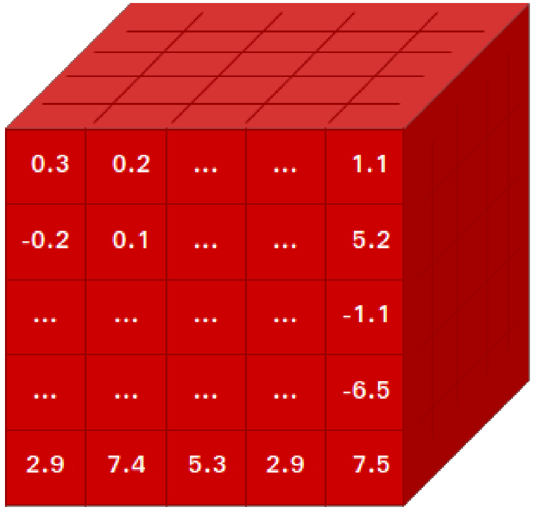


tensors

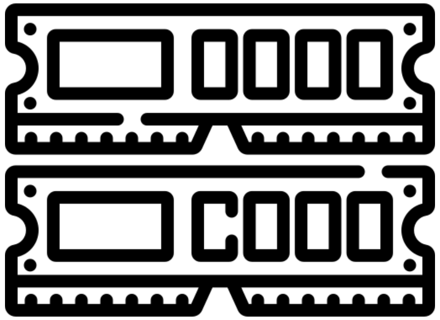


Concepts

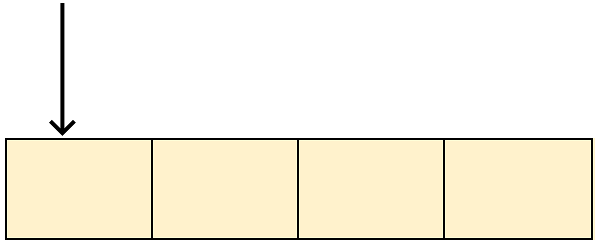
Logical
(view)



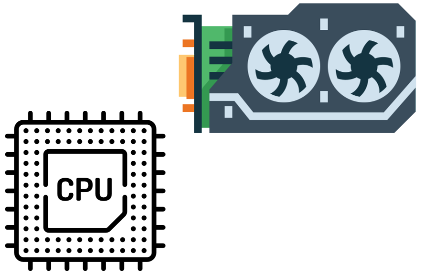
Physical
(Storage)



Stride
(Indexing)



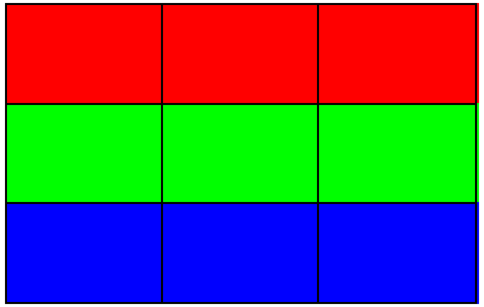
Devices



dtype

1	1.0	2	2.0

Memory Layout



Why should I Learn the internals?

Suppose we have a matrix of size $X = 1000 \times 1000$

Is transposing a costly operation?

How do you write a code to transpose? Looping?

Does accessing elements take constant time?

Is computing len(x) a costly operation?

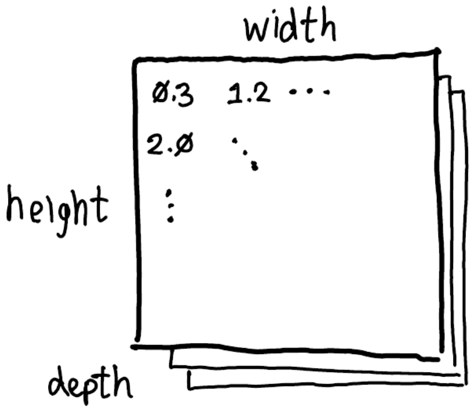
We can answer questions like these if we know how the tensors are actually stored in a hardware.

Tensor Object

Tensor
storage
stride
shape
device
size
grad
grad_fn
ndim

Some useful/important attributes of a pytorch tensor

Tensor



sizes	(D, H, W)
strides	(H*W, W, 1) contiguous
dtype	float
device	cuda:0
layout	strided

Tensor: Strided Representation

logical

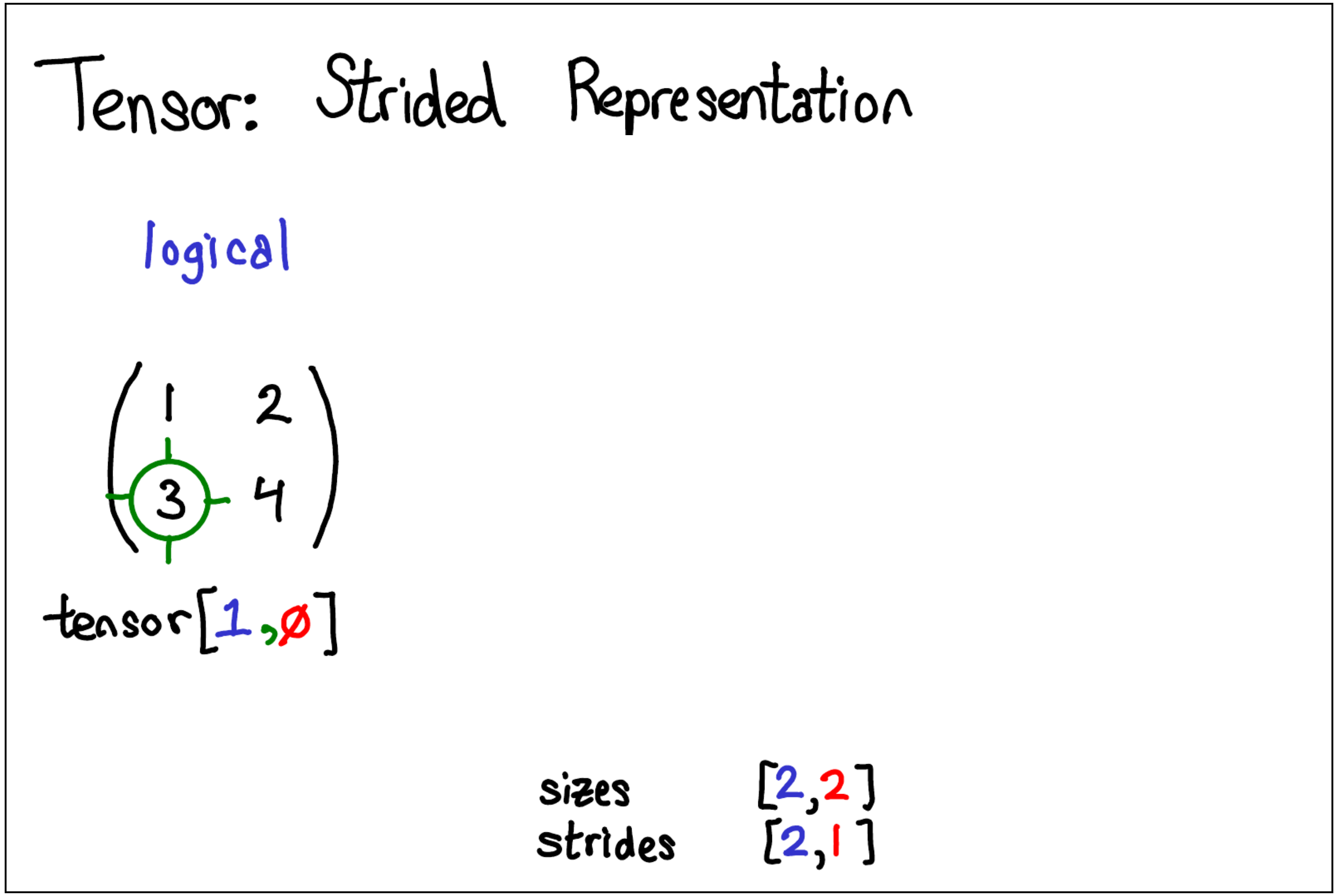
Maping follows a row-major form

maps to

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

dtype=torch.int32

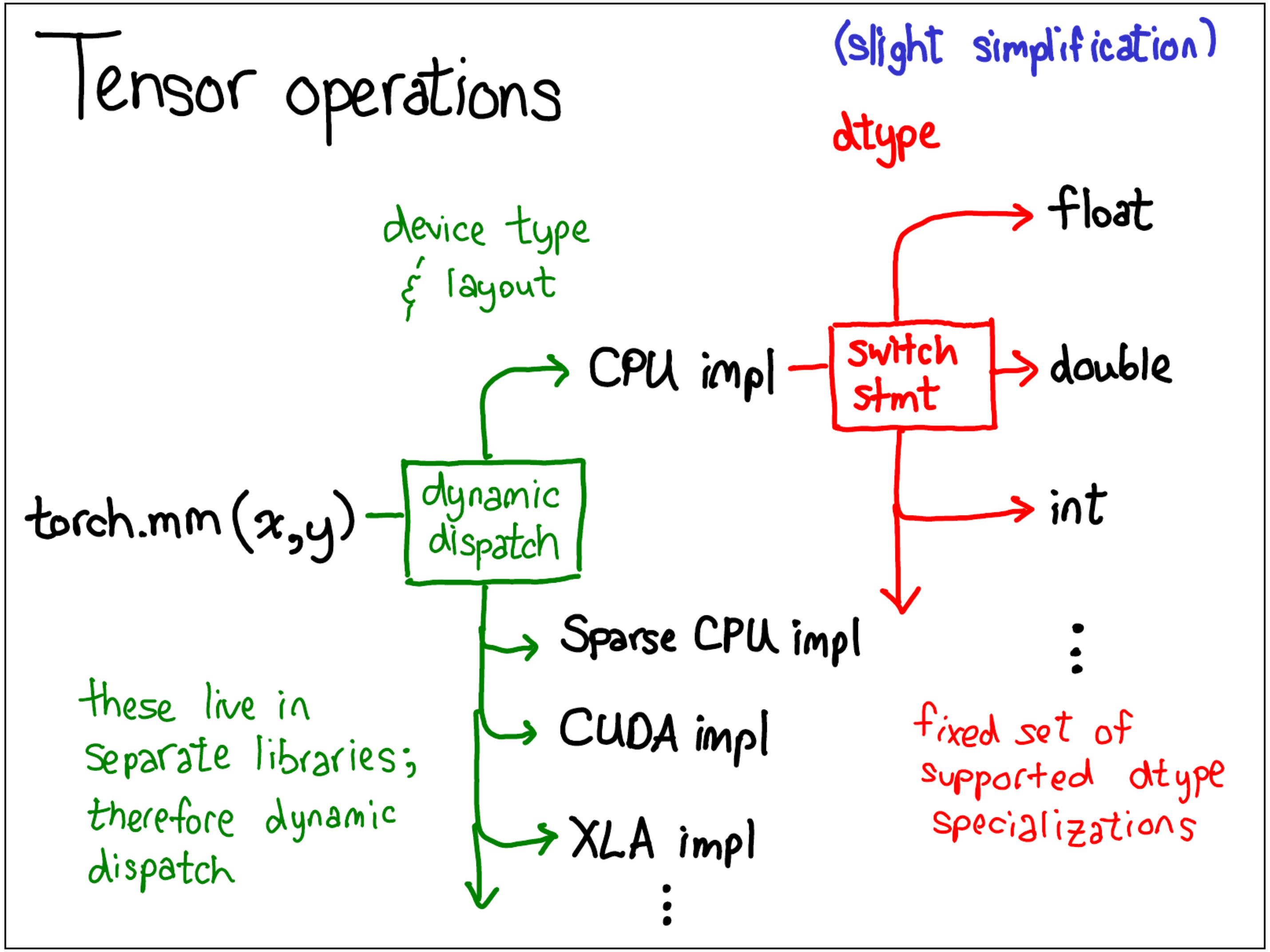
Source:http://blog.ezyang.com/2019/05/pytorch-internals/



The other representation is sparse representation

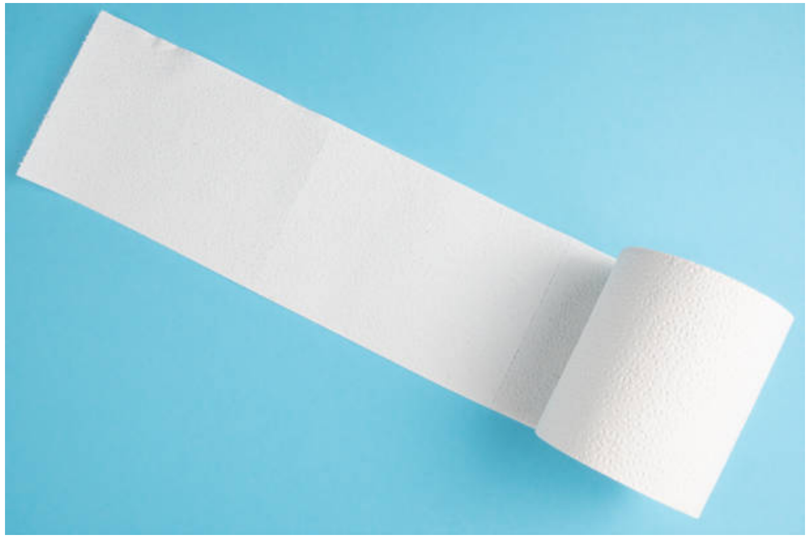
Source:http://blog.ezyang.com/2019/05/pytorch-internals/

Dispatching



Source:http://blog.ezyang.com/2019/05/pytorch-internals/

Physical storage



Source:istock

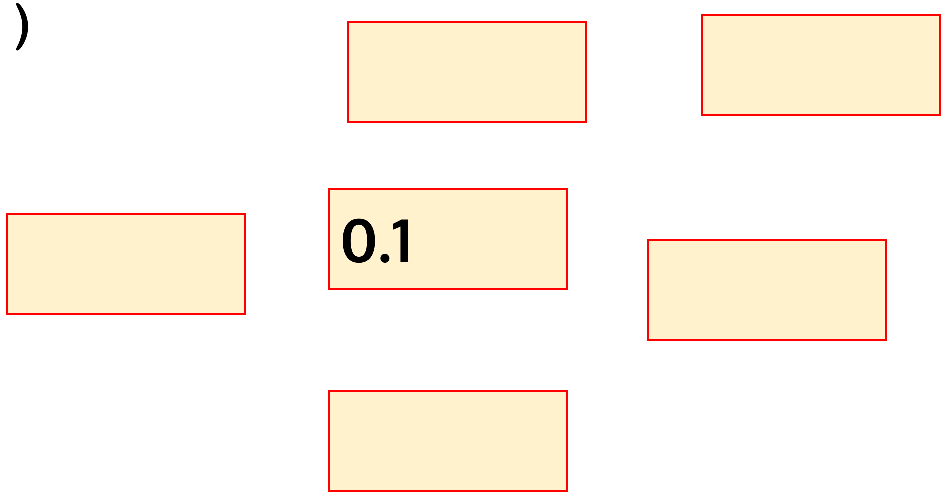
Logical View

Source:istock

Dimensions/axis/coordinate

Dim: 0

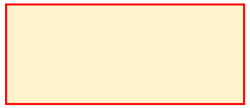
```
x = torch. Tensor(0.1)
```




```
x[0]
```

```
invalid index of a 0-dim tensor
```

```
x.item()
```



Memory location

Dimensions/axis/coordinate

Dim: 1

```
x = torch. Tensor([0.1,0.2,0.3])
```

0.1	0.2	0.3

Contiguous memory

```
x[0]
>>0.1
```

```
Stride: 1
```

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

```
[1*3+
0*1 = 3]
```

```
x[1]

stride: (3,1)

[d0*d0_stride + d1*d1_stride]
```

It is alright to view this as a matrix but not always helpful when we deal with high dim tensors

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])

torch.sum(x,dim=0)
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

1.2		

shape: (3,3)   Range:d0={0,1,2}   Range:d1={0,1,2}

stride: (3,1) [d0*d0_stride + d1*d1_stride]

since sum is across dim:0, vary dim:0 to its range (inner loop) and then dim:1 (outer loop)

0*3+ 0*1=0, x[0]=0.1

1*3+ 0*1=3, x[3]=0.4

2*3+ 0*1=6, x[6]=0.7

$$\sum = 1.2$$

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])
torch.sum(x,dim=0)
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

1.2	1.5	

```
shape: (3,3)    Range:d0={0,1,2}    Range:d1={0,1,2}
stride: (3,1)  [d0*d0_stride + d1*d1_stride]
```

```
0*3+ 1*1=1, x[1]=0.2
1*3+ 1*1=4, x[4]=0.5
2*3+ 1*1=7, x[7]=0.8
```

$$\sum = 1.5$$

since sum is across dim:0, vary dim:0 to its range (inner loop) and then dim:1 (outer loop)

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])

torch.sum(x,dim=0)
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

1.2	1.5	1.8

shape: (3,3)   Range:d0={0,1,2}   Range:d1={0,1,2}

stride: (3,1) [d0*d0_stride + d1*d1_stride]

0*3+ **2***1=2, x[1]=0.3

1*3+ **2***1=5, x[4]=0.6

2*3+ **2***1=8, x[7]=0.9

$$\sum = 1.8$$

since sum is across dim:0, vary dim:0 to its range (inner loop) and then dim:1 (outer loop)

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])

torch.sum(x,dim=1)
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

0.6		

shape: (3,3)   Range:d0={0,1,2}   Range:d1={0,1,2}

stride: (3,1) [d0*d0_stride + d1*d1_stride]

since sum is across **dim:1 now**, vary dim:1 to its range (inner loop) and then dim:1 (outer loop)

0*3+ 0*1=0, x[0]=0.1

0*3+ 1*1=1, x[1]=0.2

0*3+ 2*1=2, x[2]=0.3

$$\sum = 0.6$$

Dimensions/axis/coordinate

Dim: 2

```
x = torch. Tensor([[0.1,0.2,0.3],[0.4,0.5,0.6],[0.7,0.8,0.9]])
torch.sum(x,dim=1)
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

0.6	1.5	

shape: (3,3)   Range:d0={0,1,2}   Range:d1={0,1,2}

stride: (3,1) [d0*d0_stride + d1*d1_stride]

since sum is across **dim:1 now**, vary dim:1 to its range (inner loop) and then dim:1 (outer loop)

1*3+ 0*1=3, x[3]=0.4

1*3+ 1*1=1, x[4]=0.5

1*3+ 2*1=2, x[5]=0.6

$$\sum = 1.5$$

Dimensions/axis/coordinate

Dim: 3

```
x = torch.tensor([[[0.1,0.2],[0.3,0.4]],[[0.5,0.6],[0.7,0.8]]])

torch.sum(x,dim=1)
```

```
tensor([[[0.1000, 0.2000],
         [0.3000, 0.4000]],

        [[0.5000, 0.6000],
         [0.7000, 0.8000]]])
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8

Let's figure out the shape of the tensor by starting with the right most dimension

$d_k = 2$ (because there are two numbers (scalars) enclosed by a square brackets

$d_{k-1} = 2$ (because there are two vectors (dim:1) enclosed by a square brackets

$d_{k-2} = 1$

Dimensions/axis/coordinate

Dim: 3

```
x = torch.tensor([[[0.1,0.2],[0.3,0.4]],[[0.5,0.6],[0.7,0.8]]])

torch.sum(x,dim=1)
```

```
tensor([[[0.1000, 0.2000],
         [0.3000, 0.4000]],

        [[0.5000, 0.6000],
         [0.7000, 0.8000]]])
```

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8

```
shape: (1,2,2)    Range:d0={0,1,2}    Range:d1={0,1,2}

stride: (4,2,1)   [d0*d0_stride + d1*d1_stride+d2*d2_stride]
```

$x: 2 \times 2 \times 3 \times 2$

```
tensor([[[[0, 2],
          [1, 1],
          [0, 2]],

         [[1, 2],
          [1, 2],
          [1, 1]]],


        [[[2, 0],
          [0, 1],
          [2, 1]],

         [[0, 1],
          [1, 2],
          [1, 2]]]])
```

`torch.sum(x,dim=2)`

`Range:d0={0,1}` `Range:d2={0,1,2}`

`Range:d1={0,1}` `Range:d3={0,1}`

`stride: (12,6,2,1)`

d0*12	d1*6	d2*2	d3*1
0	0	0	0
		1	
		2	

`index: 0+0+0+0=0, x[0]=0`

`index: 0+0+2+0=0, x[2]=1`

`index: 0+0+4+0=0, x[4]=0`

$\sum = 1$

$x : 2 \times 2 \times 3 \times 2$

```
tensor([[[[0, 2],
          [1, 1],
          [0, 2]],

         [[1, 2],
          [1, 2],
          [1, 1]]],


        [[[2, 0],
          [0, 1],
          [2, 1]],

         [[0, 1],
          [1, 2],
          [1, 2]]]])
```

```
torch.sum(x,dim=2)
```

```
Range:d0={0,1}   Range:d2={0,1,2}
Range:d1={0,1}   Range:d3={0,1}
stride: (12,6,2,1)
```

d0*12	d1*6	d2*2	d3*1
0	0	0	
		1	1
		2	

```
index: 0+0+0+1=1, x[1]=2
index: 0+0+2+1=3, x[3]=1
index: 0+0+4+1=0, x[5]=2
```

$\sum = 5$

Move into the right adjacent dimension

$x: 2 \times 2 \times 3 \times 2$

```
tensor([[[[0, 2],
          [1, 1],
          [0, 2]],

         [[1, 2],
          [1, 2],
          [1, 1]]],


        [[[2, 0],
          [0, 1],
          [2, 1]],

         [[0, 1],
          [1, 2],
          [1, 2]]]])
```

`torch.sum(x,dim=2)`

`Range:d0={0,1}` `Range:d2={0,1,2}`

`Range:d1={0,1}` `Range:d3={0,1}`

`stride: (12,6,2,1)`

d0*12	d1*6	d2*2	d3*1
0		0	0
	1	1	
		2	

`index: 0+6+0+0=6, x[6]=1`

`index: 0+6+2+0=8, x[8]=1`

`index: 0+6+4+0=10, x[10]=1`

$\sum = 3$

```
tensor([[[1, 5],
         [3, 5]],

        [[4, 2],
         [2, 5]]])
```

Move into left adjacent dimension

$x : 2 \times 2 \times 3 \times 2$

```
tensor([[[[0, 2],
          [1, 1],
          [0, 2]],

         [[1, 2],
          [1, 2],
          [1, 1]]],


        [[[2, 0],
          [0, 1],
          [2, 1]],

         [[0, 1],
          [1, 2],
          [1, 2]]]])
```

We call the 'sum' a **reduction operation** as it reduces the dim from 3 to 1.

$$2 \times 2 \times 3 \times 2$$

$2 \times 2 \times 2$

```
tensor([[[1, 5],
         [3, 5]],

        [[4, 2],
         [2, 5]]])
```

`torch.sum(x,dim=2)`

$2 \times 3 \times 2$

```
tensor([[[1, 4],
         [2, 3],
         [1, 3]],

        [[2, 1],
         [1, 3],
         [3, 3]]])
```

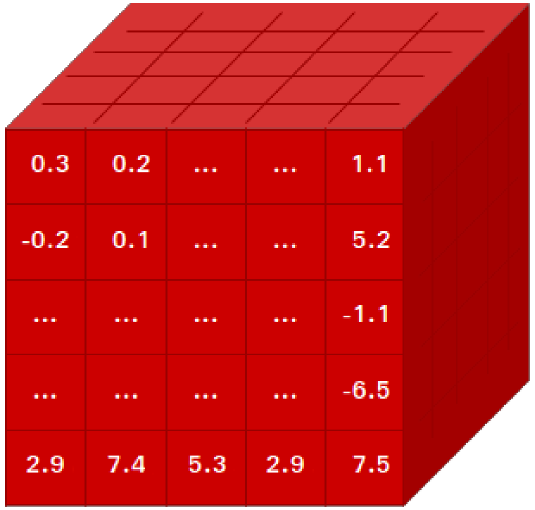
`torch.sum(x,dim=1)`

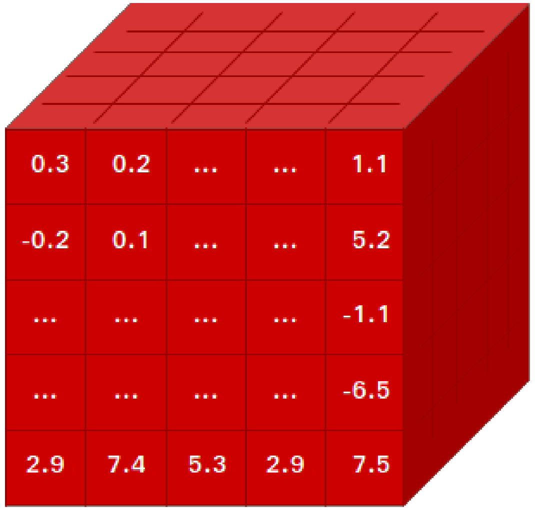
$2 \times 2 \times 3$

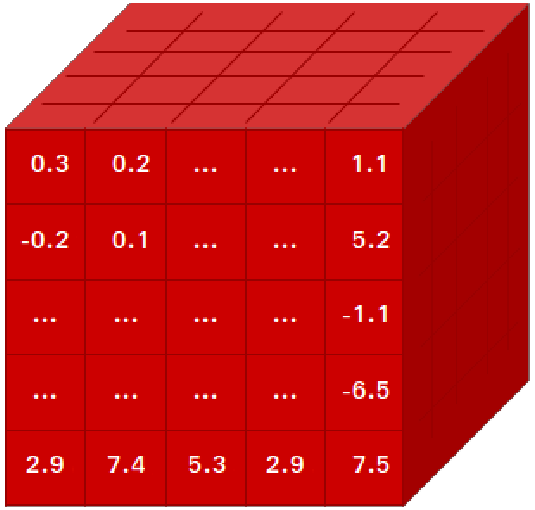
```
tensor([[[2, 2, 2],
         [3, 3, 2]],

        [[2, 1, 3],
         [1, 3, 3]]])
```

`torch.sum(x,dim=3)`

All these cubes are the elements at 0-th dim of a tensor of shape $3 \times 5 \times 5 \times 5$. The first number 3 denotes three elements in zeroth dim and each of size 5 x 5 x 5 and







$$3 \times 5 \times 5 \times 5$$

Let's switch to Colab Notebook