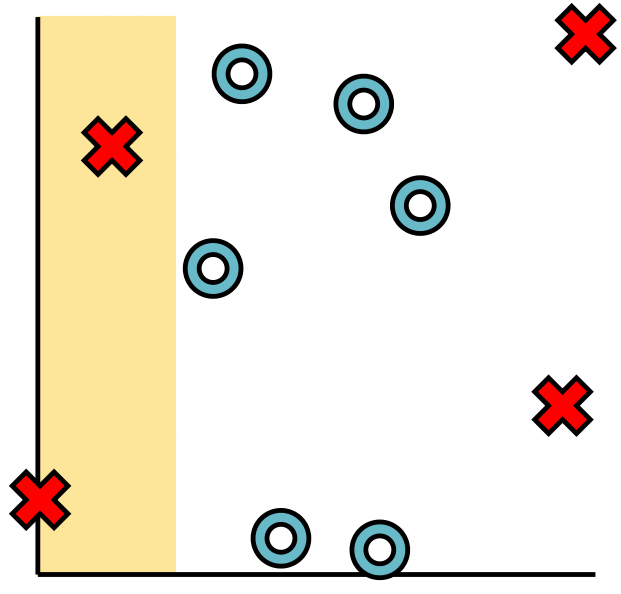
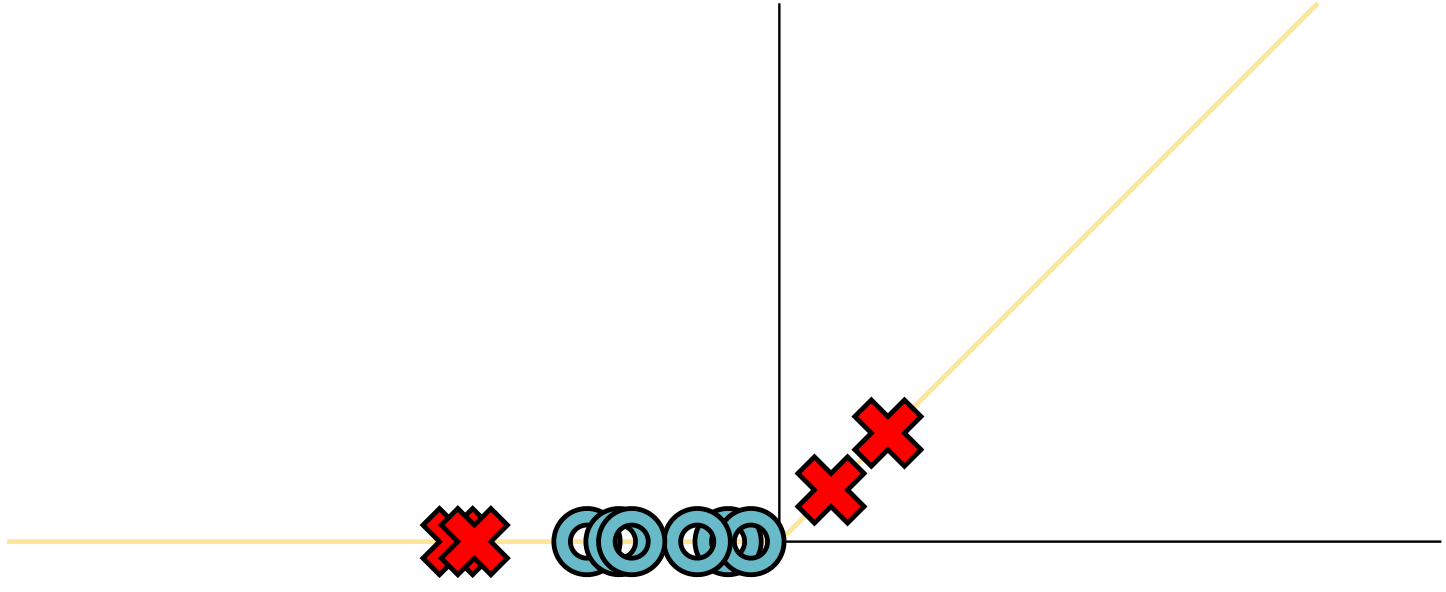
Module 2.1 - Tensors

Intuition: Split 1



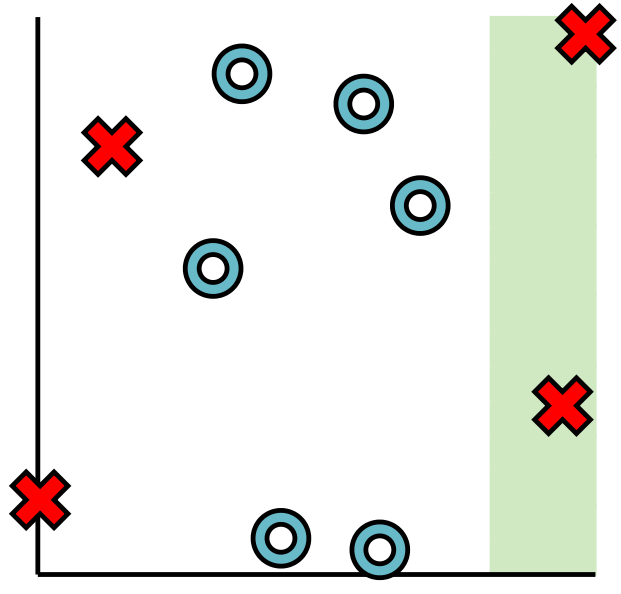
Reshape: ReLU



Math View

$$h_1 = \mathrm{ReLU}(\mathrm{lin}(x; w^0, b^0))$$

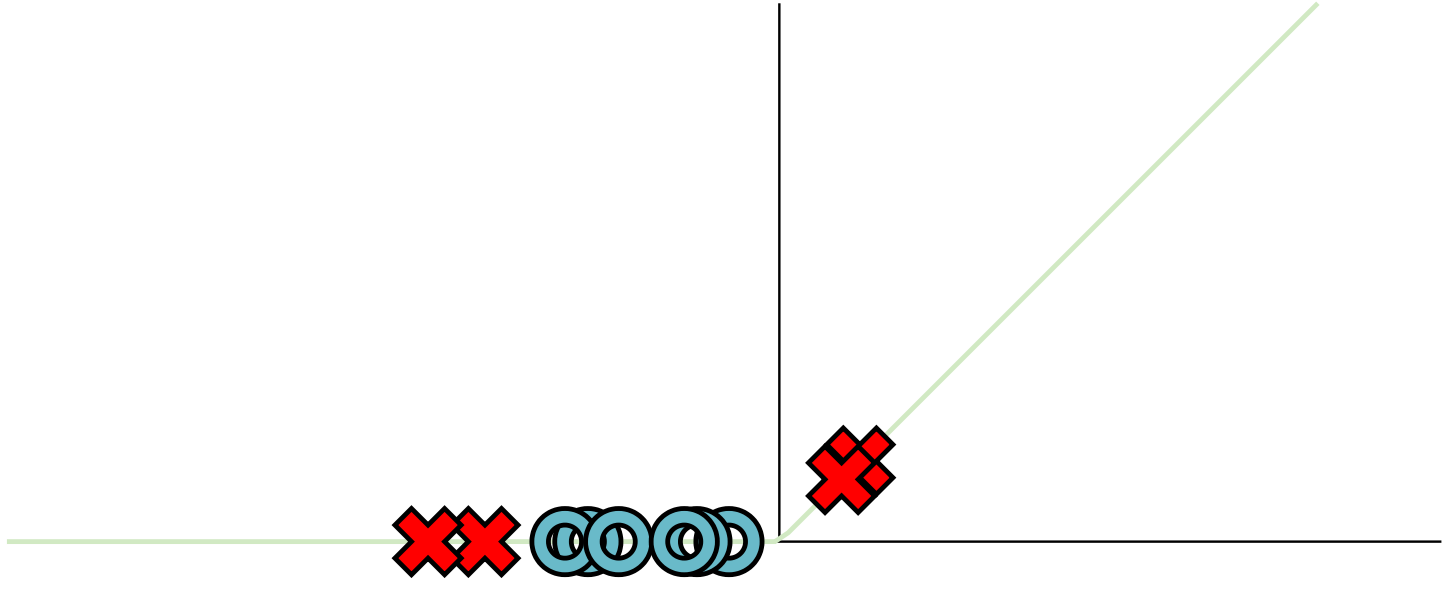
Intuition: Split 2



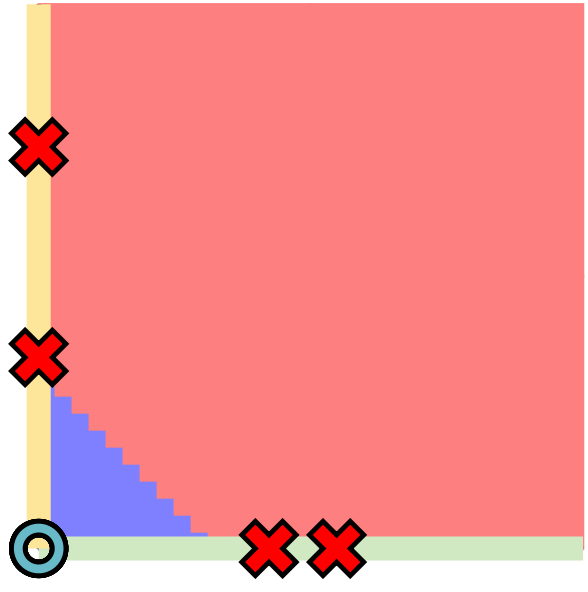
Math View

$$h_2 = \mathrm{ReLU}(\mathrm{lin}(x; w^1, b^1))$$

Reshape: ReLU



Reshape: ReLU



Math View (Alt)

$$
\begin{aligned}
\text{lin}(x; w, b) &= x_1 \times w_1 + x_2 \times w_2 + b \\
h_1 &= \text{ReLU}(\text{lin}(x; w^0, b^0)) \\
h_2 &= \text{ReLU}(\text{lin}(x; w^1, b^1)) \\
m(x_1, x_2) &= \text{lin}(h; w, b)
\end{aligned}
$$

Code View

Model

```
class Network(minitorch.Module):
    def __init__(self):
        super().__init__()
        self.unit1 = LinearModule()
        self.unit2 = LinearModule()
        self.classify = LinearModule()

    def forward(self, x):
        # yellow
        h1 = self.unit1.forward(x).relu()
        # green
        h2 = self.unit2.forward(x).relu()
        return self.classify.forward((h1, h2))
```

Quiz

Outline

- Tensors
- Operations
- Strides

Tensors

Motivation

$$
\begin{aligned}
\text{lin}(x; w, b) &= x_1 \times w_1 + x_2 \times w_2 + b \\
h_1 &= \text{ReLU}(\text{lin}(x; w^0, b^0)) \\
h_2 &= \text{ReLU}(\text{lin}(x; w^1, b^1)) \\
m(x_1, x_2) &= \text{lin}(h; w, b)
\end{aligned}
$$

Parameters: $w_1, w_2, w_1^0, w_2^0, w_1^1, w_2^1, b, b^0, b^1$

- This is really messy!

Matrix Form

$$\mathbf{h} = \mathrm{ReLU}(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)})$$
$$m(\mathbf{x}) = \mathbf{W}\mathbf{h} + \mathbf{b}$$

Parameters: $\mathbf{W}, \mathbf{b}, \mathbf{W}^{(0)}, \mathbf{b}^{(0)}$

- Matrix - compute a bunch of linears at once (may be more than 2!)

Matrix / Tensors

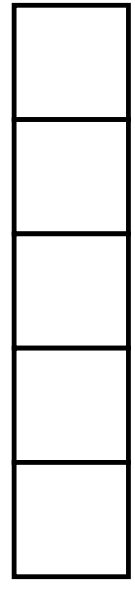
- Multi-dimensional arrays
- Basis for an mathmatical programming
- Similar foundation for many libraries (matlab, numpy, etc)

Terminology

- 0-Dimensional Scalar
- `Scalar` from module-0

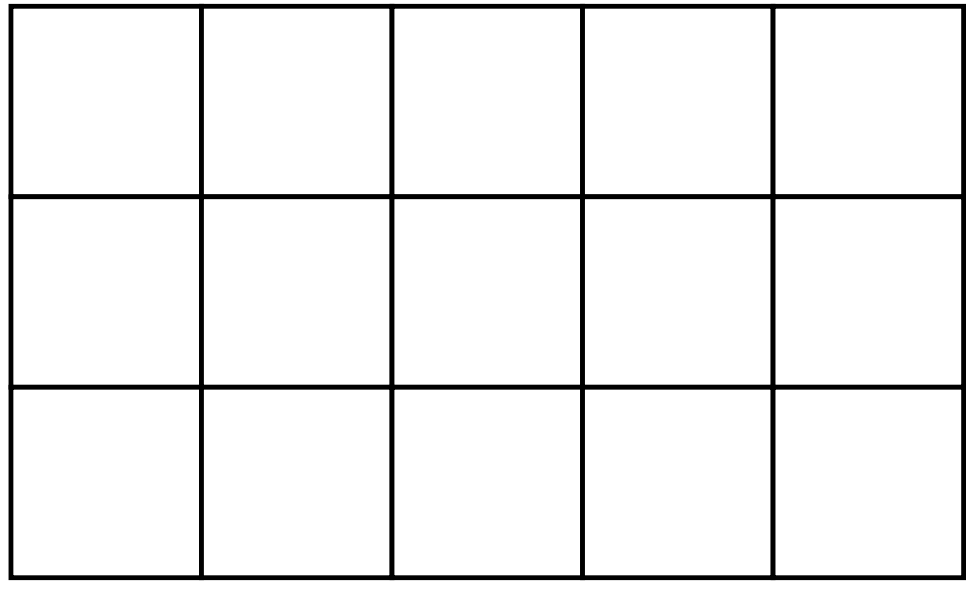
Terminology

- 1-Dimensional - Vector



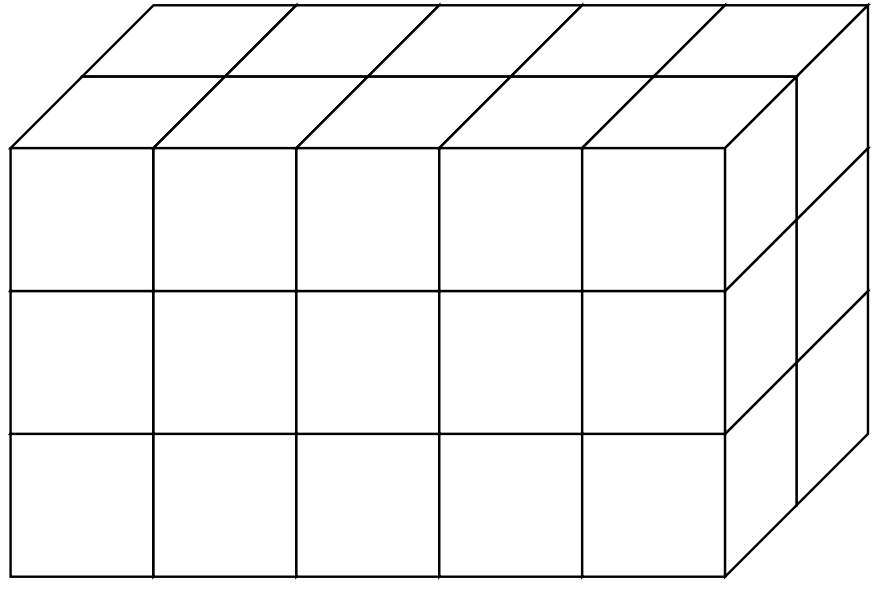
Terminology

- 2-Dimensional - Matrix



Terminology

- n-dimensions - Tensor



Terminology

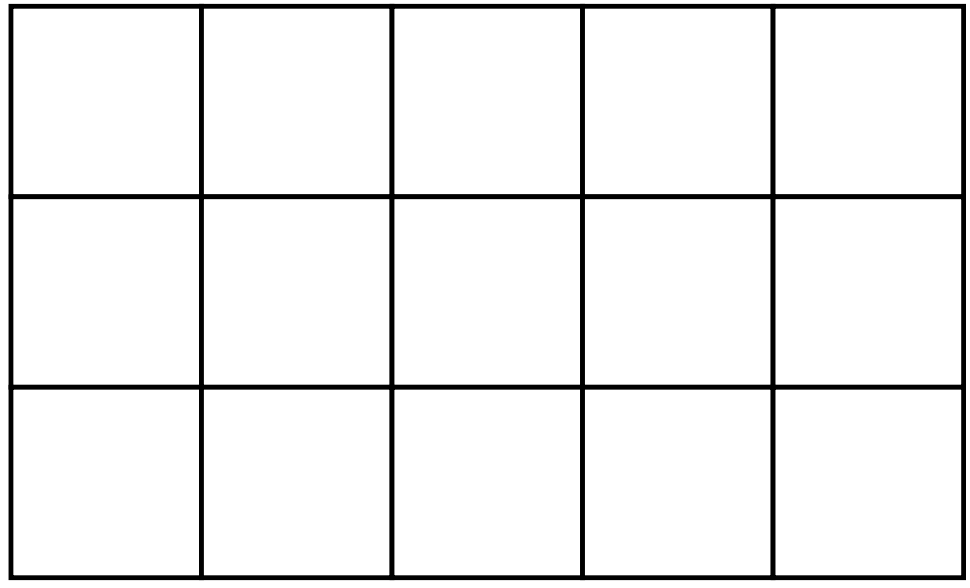
- Dims - # dimensions (`x.dims`)
- Shape - # cells per dimension (`x.shape`)
- Size - # cells (`x.size`)

Visual Convention

- depth
- row
- columns

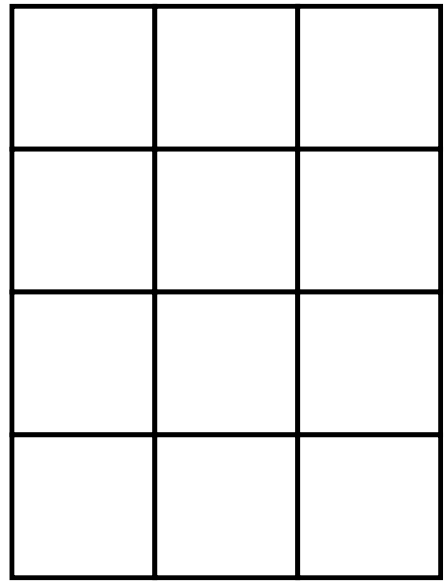
Example

- dims: 2
- shape: (3, 5)
- size : 15



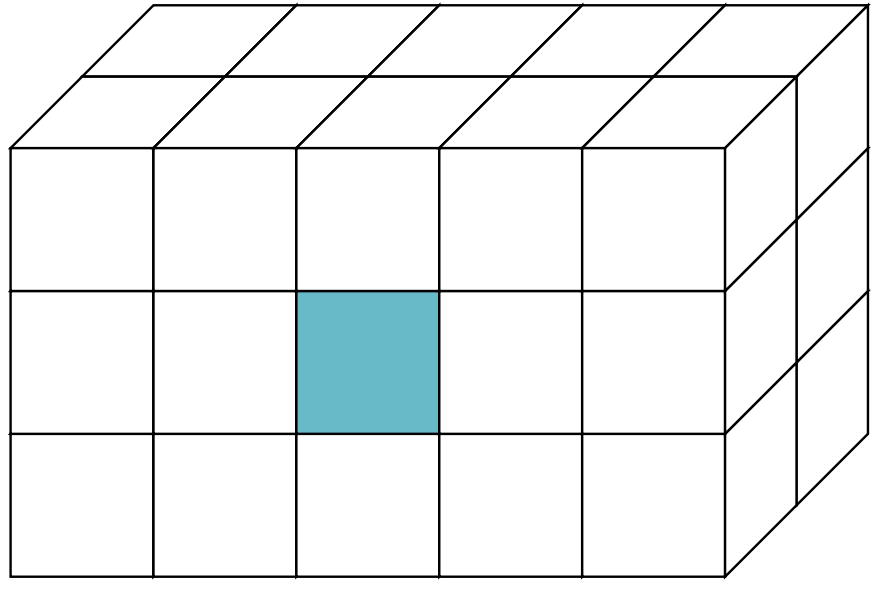
Example

- dims: ?
- shape: ?
- size : ?



Indexing

- Indexing syntax: `x[0, 1, 2]`



Implementing Tensors

Why not just use lists?

- Functions to manipulate shape
- Mathematical notation
- Enables autodiff
- Efficient control of memory (Module-3)

Tensor Usage

Unary

```
new_tensor = x.log()
```

Binary (for now, only same shape)

```
new_tensor = x + x
```

Reductions

```
new_tensor = x.sum()
```

Immutable Operations

- We never change the tensors itself (mostly)
- All operations return a new tensor (just like `Scalar``)

What's bad about tensors?

- Hard to grow or shrink
- Only numerical values
- Lose comprehensions / python built-ins
- Shapes are easy to mess up

Next Couple Lectures

- No autodifferentiation for now
- Only consider forward tensor operations
- Add autodiff afterwards

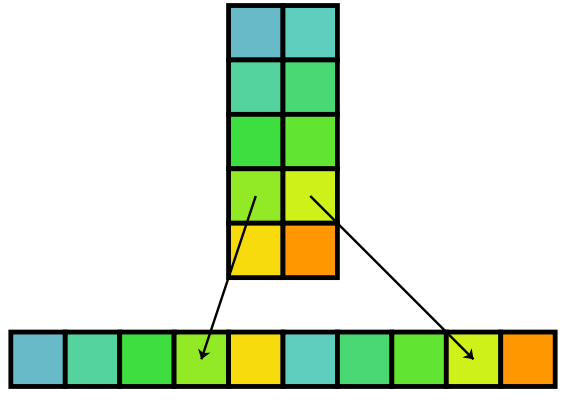
Tensor Internals

How does this work

- **Storage** : 1-D array of numbers of length `size`
- **Strides** : tuple that provides the mapping from user `indexing` to the `position` in the 1-D `storage`.

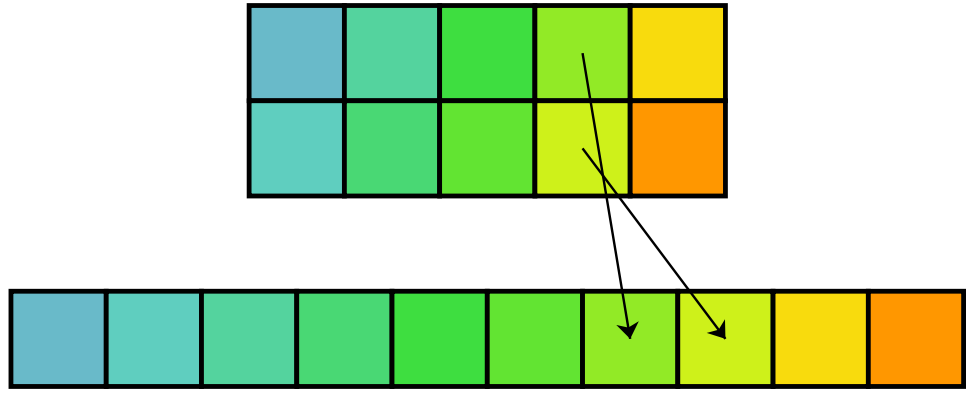
Strides

- Stride: $(1, 5)$
- Shape: $(5, 2)$



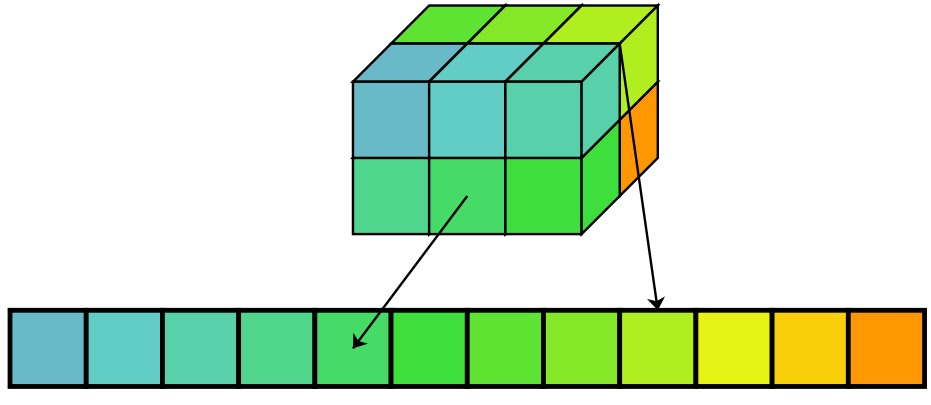
Strides

- Stride: $(1, 2)$
- Shape: $(2, 5)$



Strides

- Shape: $(2, 2, 3)$
- Stride: $(6, 3, 1)$

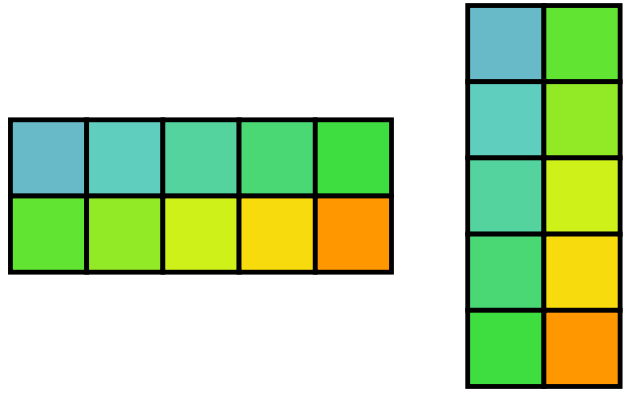


Which do we use?

- Contiguous: Bigger strides left
- (s_1, s_2, s_3)
- However, need to handle all cases.

Strides are useful: Transpose

Can transpose without copying.



Operation 1: Indexing

- $x[i, j, k]$

How to find data point?

Operation 2: Movement

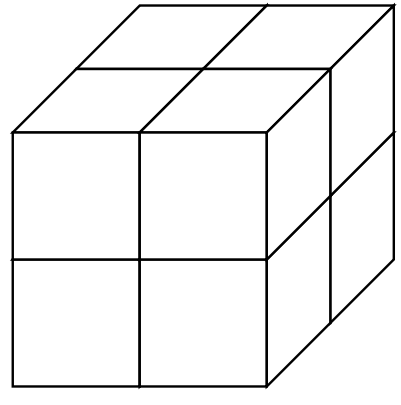
How do I move to the next in the row? Column?

Operation 3: Reverse Indexing

How do I find the index for data?

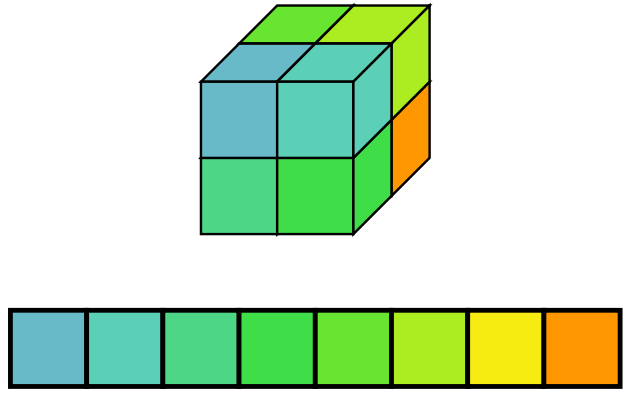
Stride Intuition

- Numerical bases,
- Index for position 0? Position 1? Position 2?



Stride Intuition

- Index for position 0? Position 1? Position 2?
- $[0, 0, 0], [0, 0, 1], [0, 1, 0]$



Conversion Formula

- Divide and mod
- $k = p$
- $j = (p // s_2)$
- ...

Implementation

- TensorData : Manager of strides and storage

Module-2

Overview

- `tensor.py` - Tensor Variable
- `tensor_functions.py` - Tensor Functions
- `tensor_data.py` - Storage and Indexing
- `tensor_ops.py` - Low-level tensor operations

Q&A