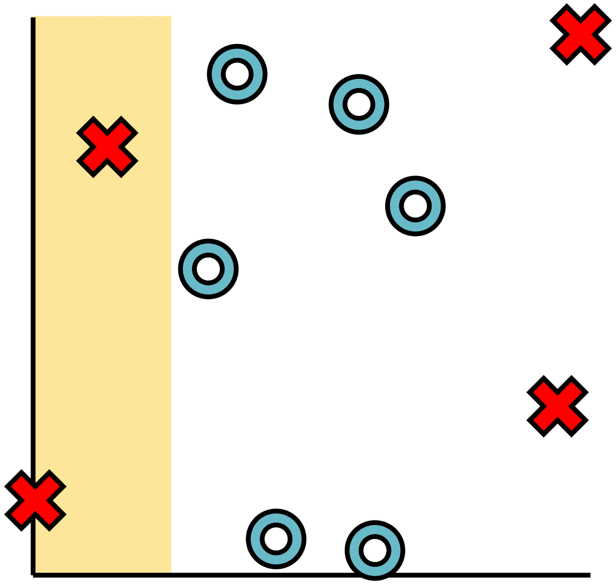
Module 2.1 - Tensors

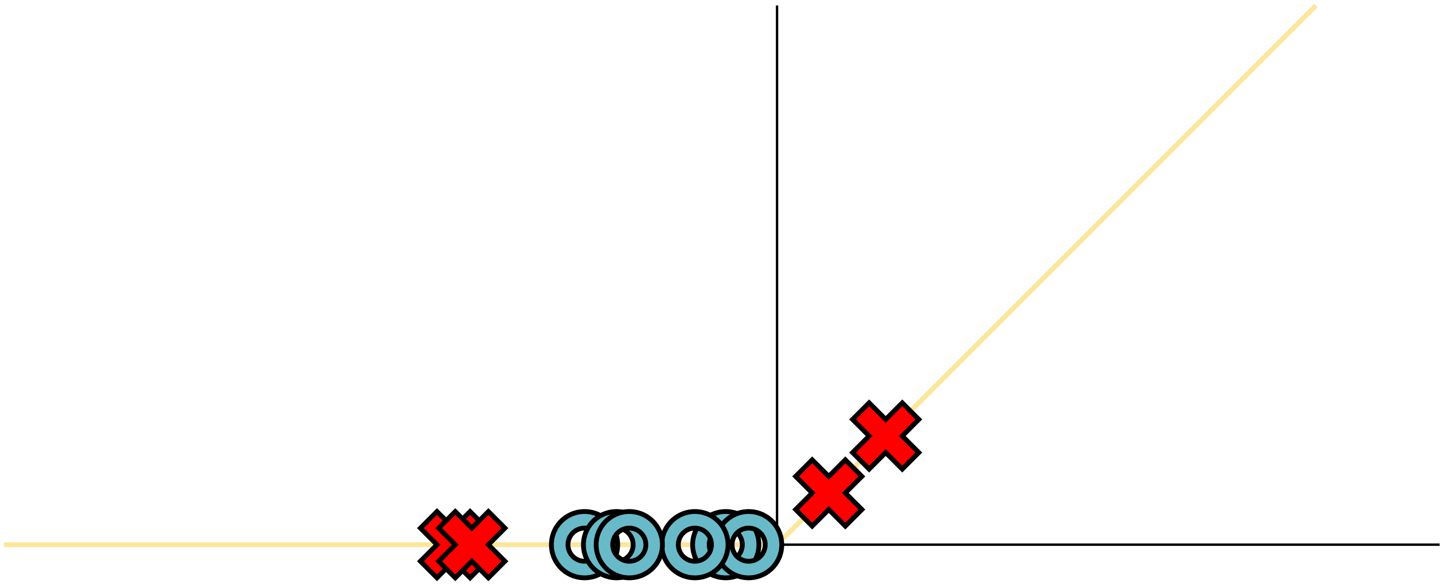
Intuition: Split 1

```
yellow = Linear(-1, 0, 0.25)
```



Loading [MathJax]/extensions/Safe.js

Reshape: ReLU

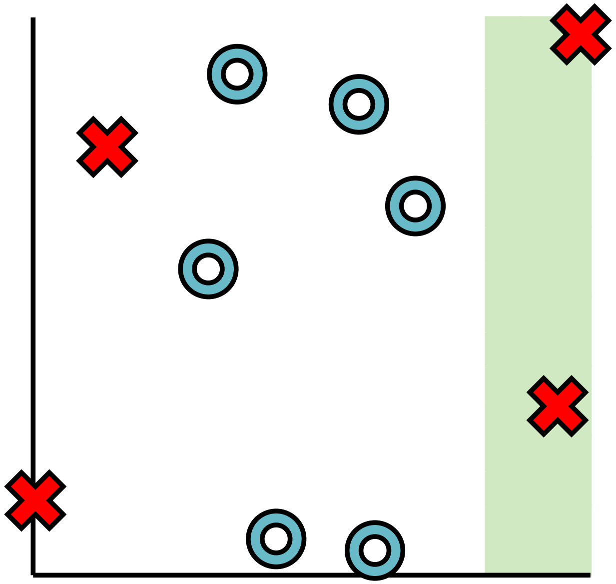


Math View

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

Intuition: Split 2

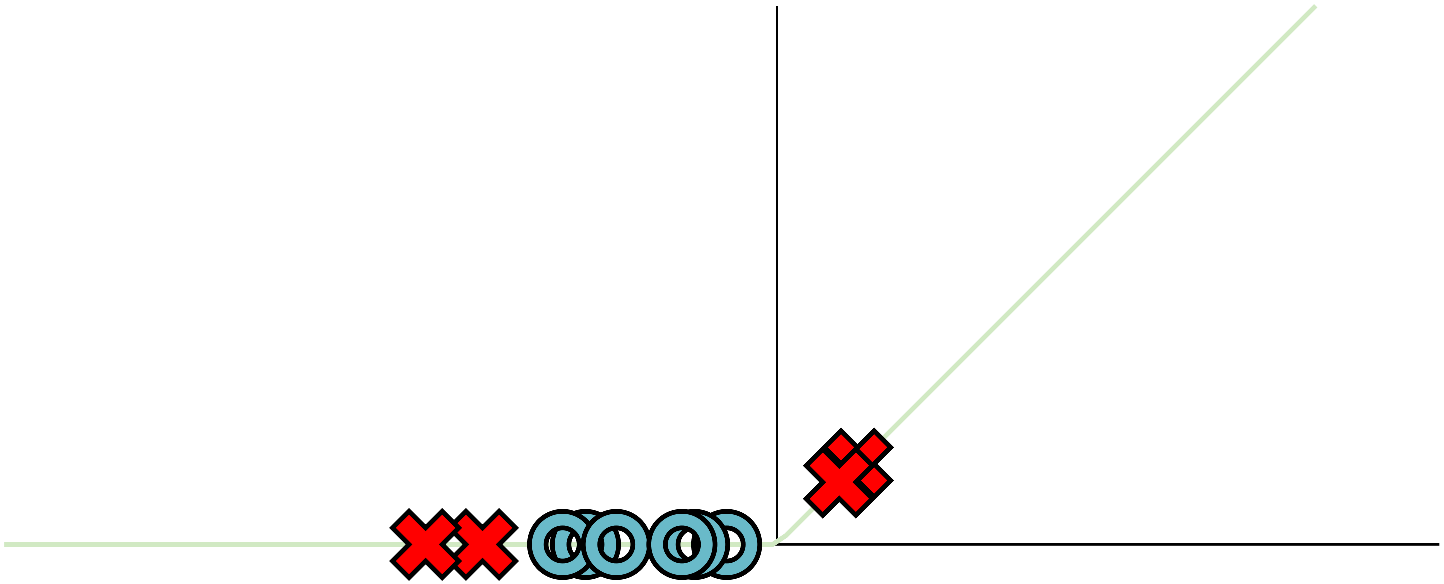
```
green = Linear(1, 0, -0.8)
```



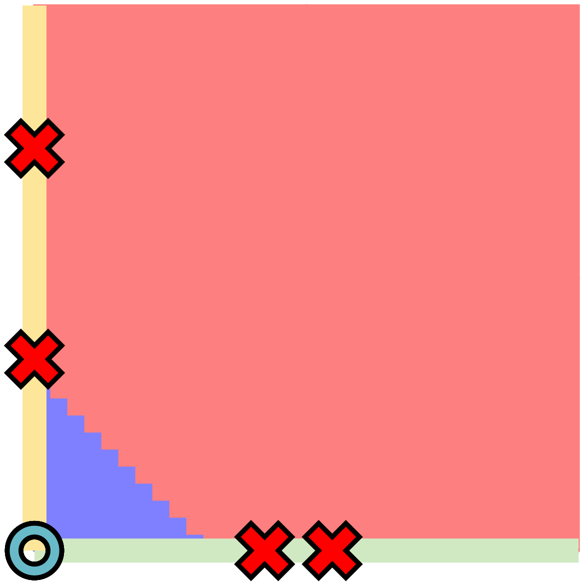
Math View

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

Reshape: ReLU



Reshape: ReLU



Math View (Alt)

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

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

Quiz

Outline

- Tensors
- Operations
- Strides

Tensors

Motivation

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

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

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

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

- Need to be careful thought

Tensors

What is it?

- Fancy multi-dimensional arrays
- Basis for an mathmatical programming

Tensors!

Key Building Block

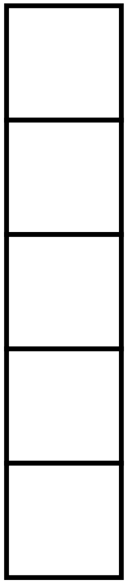
- Matlab
- Numpy
- Tensorflow, etc.

Terminology

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

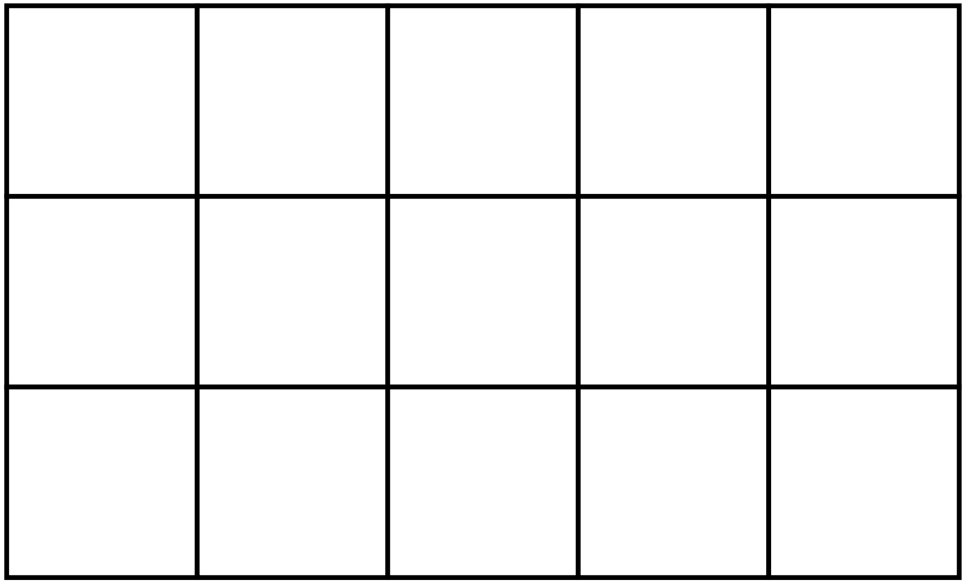
Terminology

- 1-Dimensional
- Math: Vector



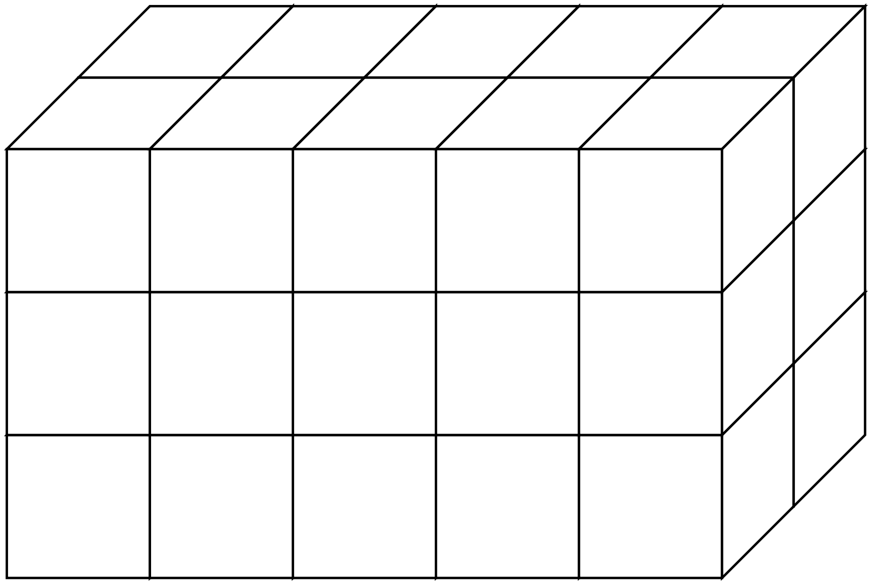
Terminology

- 2-Dimensional
- Math: Matrix



Terminology

- Arbitrary dimensions - Tensor (Array in numpy)

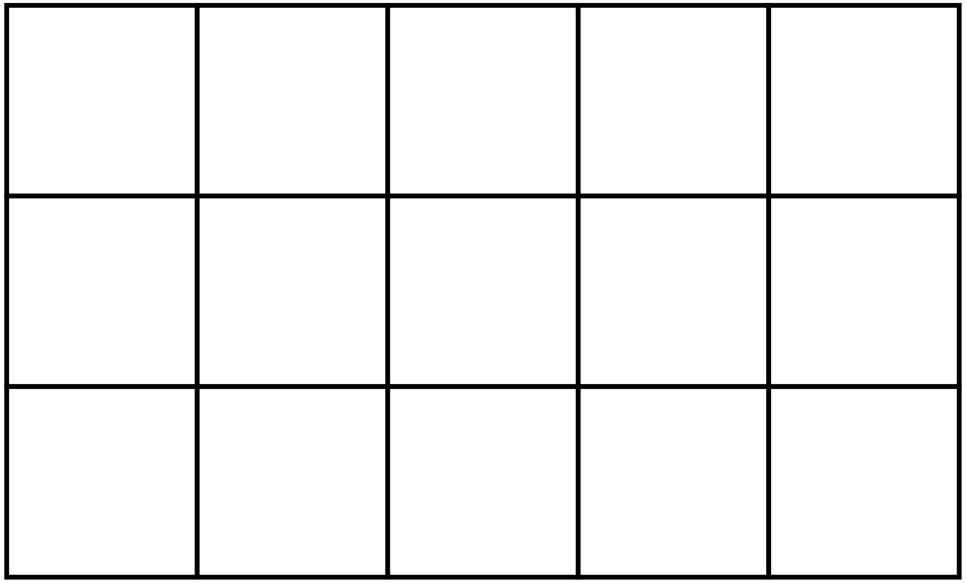


Terminology

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

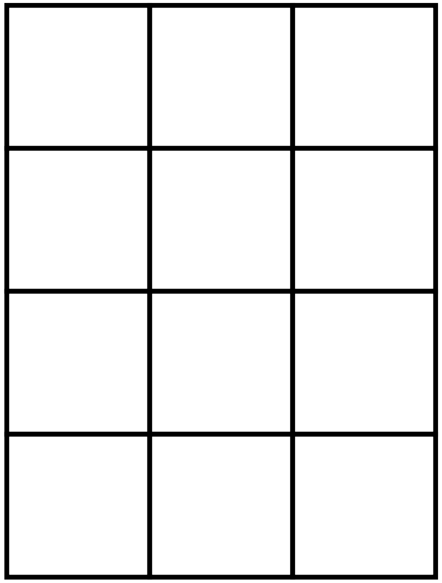
Example

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



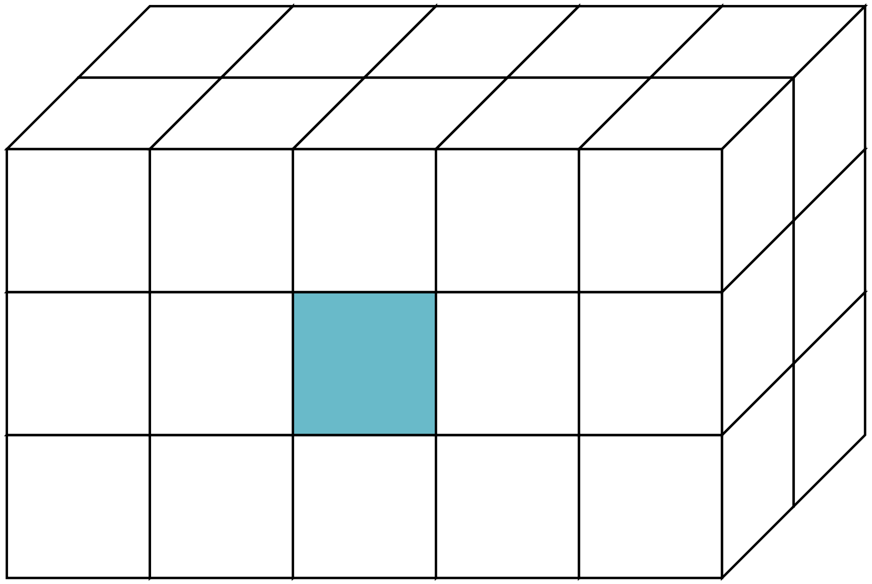
Example

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



Indexing

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



Loading [MathJax]/extensions/Safe.js

Convention

- depth
- row
- columns

Operations

Goal

- Support user api
- Keep track of tensor properties
- Setup fast / simple Functions

Tensor Usage

Unary

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

Binary (for now, only same shape)

```
new_tensor = x + x
```

Reductions

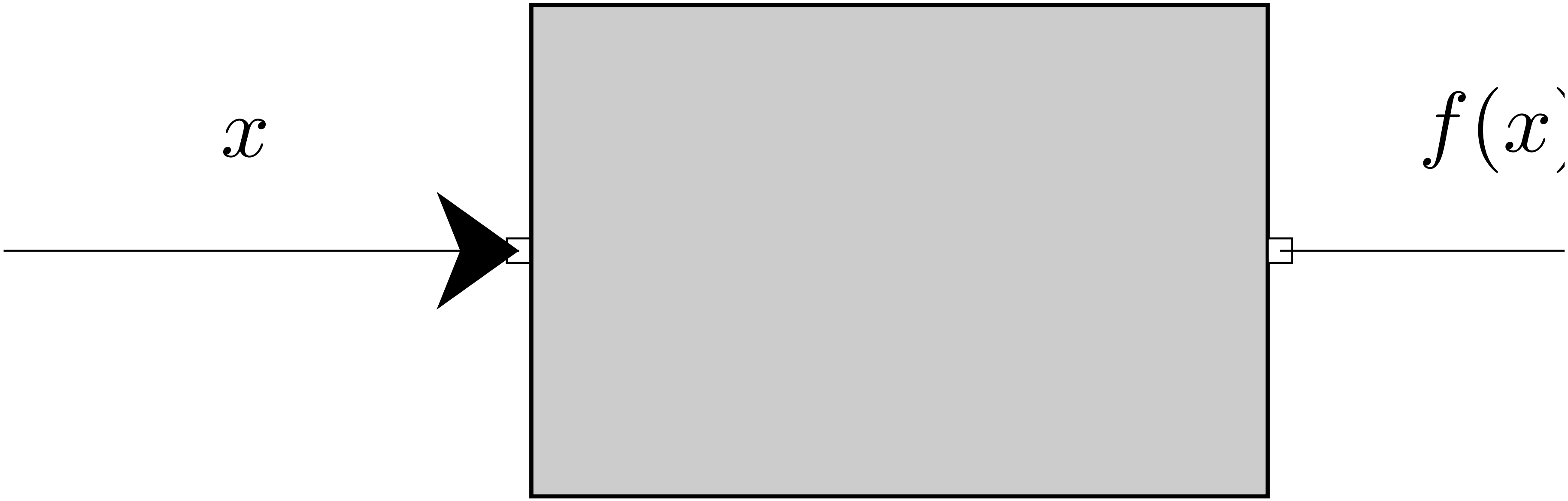
```
new_tensor = x.sum()
```

Loading [MathJax]/extensions/Safe.js

Idea 1: Immutable Operations

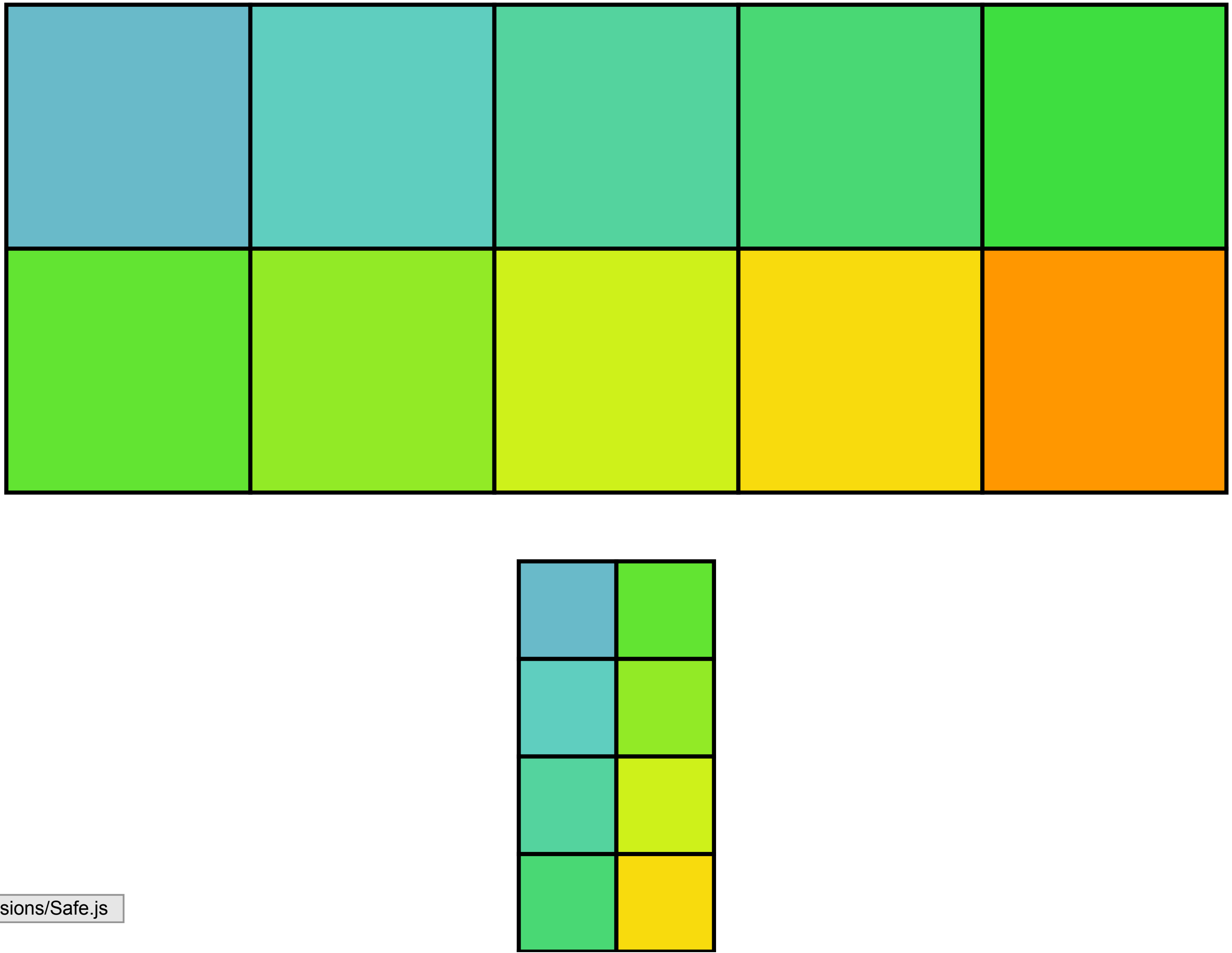
- Minitorch doesn't let you update tensors
- All operations return a new tensor (just like scalar)

```
draw_boxes(["$x$", "$f(x)$"], [1])
```

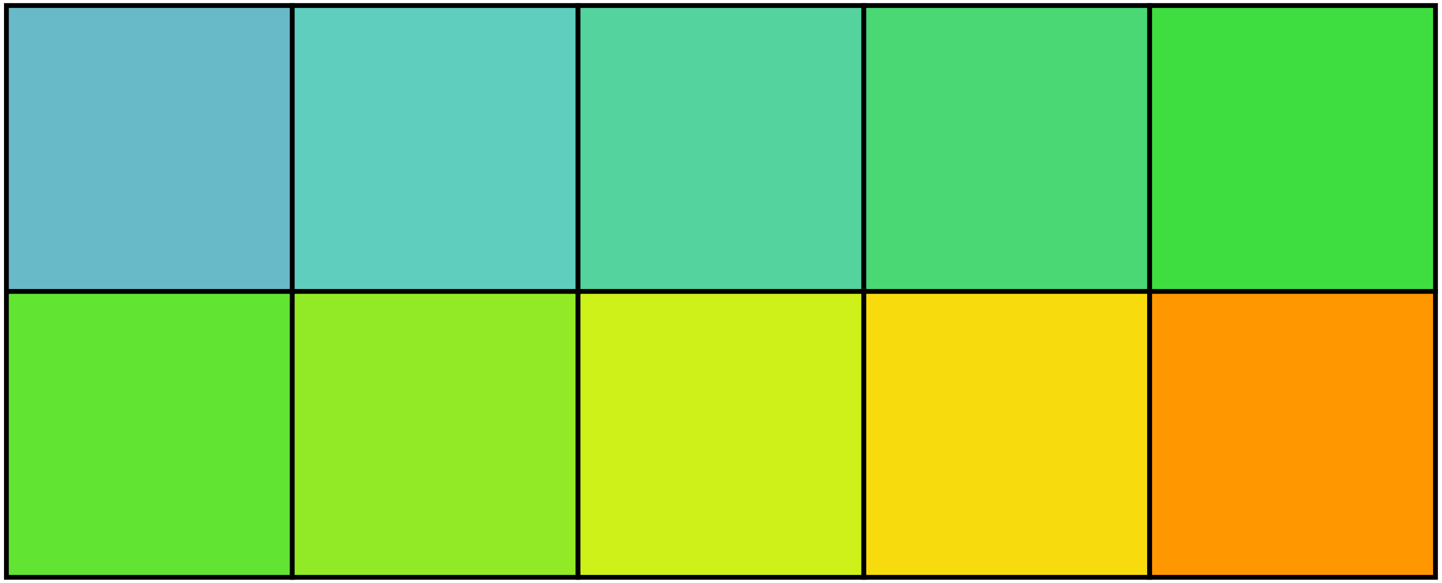


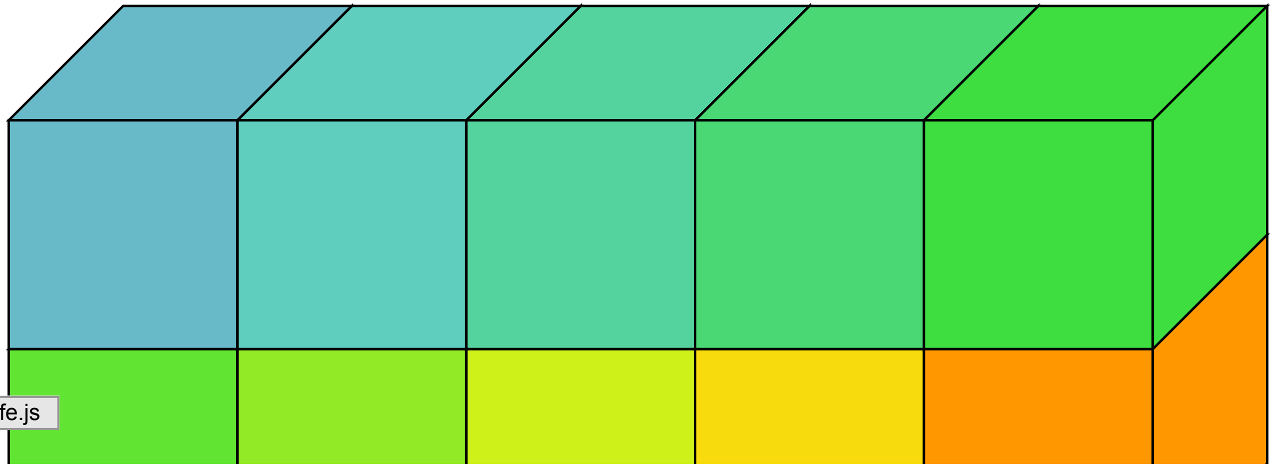
Loading [MathJax]/extensions/Safe.js

Shape Manipulation: Permutation



Shape Manipulation: View





Why not just use lists?

- Functions to manipulate shape
- Mathematical notation
- Can act as Variables / Parameters
- Efficient control of memory (Module-3)

Why not lists?

Matrix (5, 2)

```
ls = [[1, 2], [3, 4], [5, 7], [2, 3], [2, 4]]
```

View (1, 5, 2)

```
new_ls = [[[ls[j][i] for i in range(2)] for j in range(5)]]
```

Transpose (2, 5)

```
matrix_trans = [[ls[i][j] for i in range(5)] for j in range(2)]
```

Issues

- Operators requires copying

```
matrix_trans = [[ls[i][j] for i in range(5)] for j in range(2)]
```

- Storage shaped based on usage

```
new_tensor = [[[ls[j][i] for i in range(2)] for j in range(5)]]
```

Idea 2: Views

- Seperate storing information from user view
- Keep a mapping from users version to storage

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

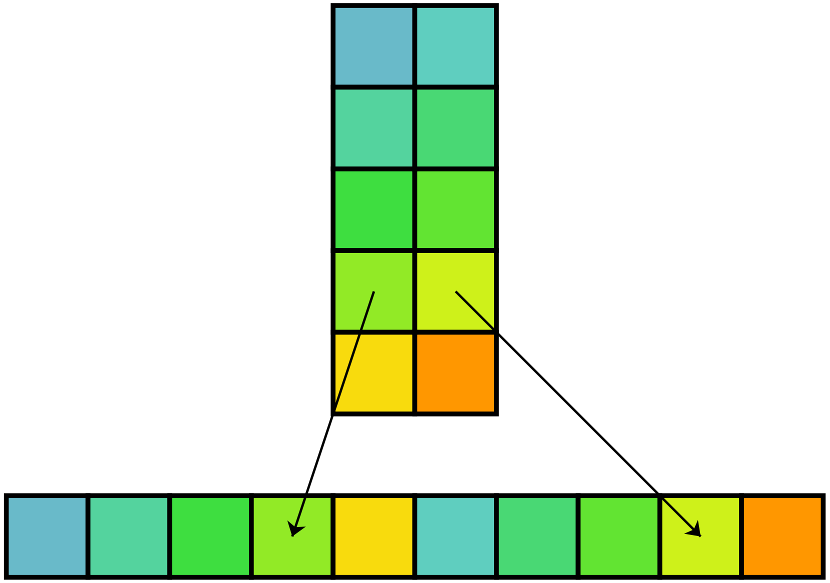
- No autodifferentiation for now
- Only consider forward tensor operations
- Add autodiff afterwards

Tensor Internals

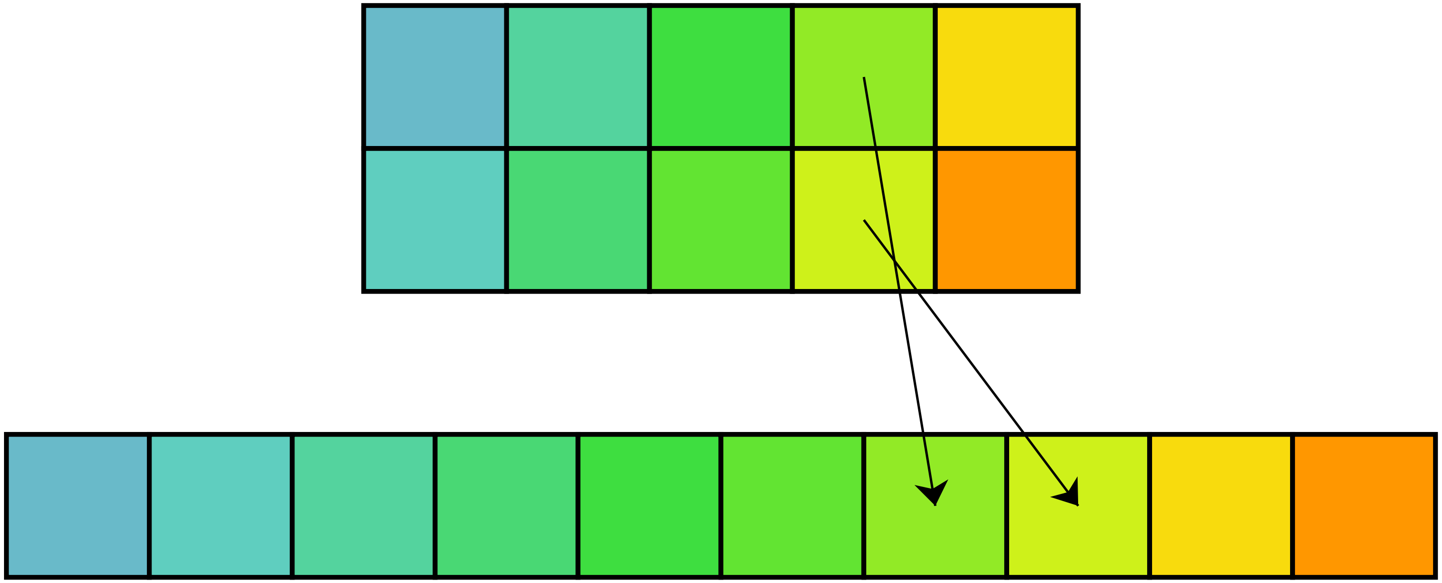
How does this work

- **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`.

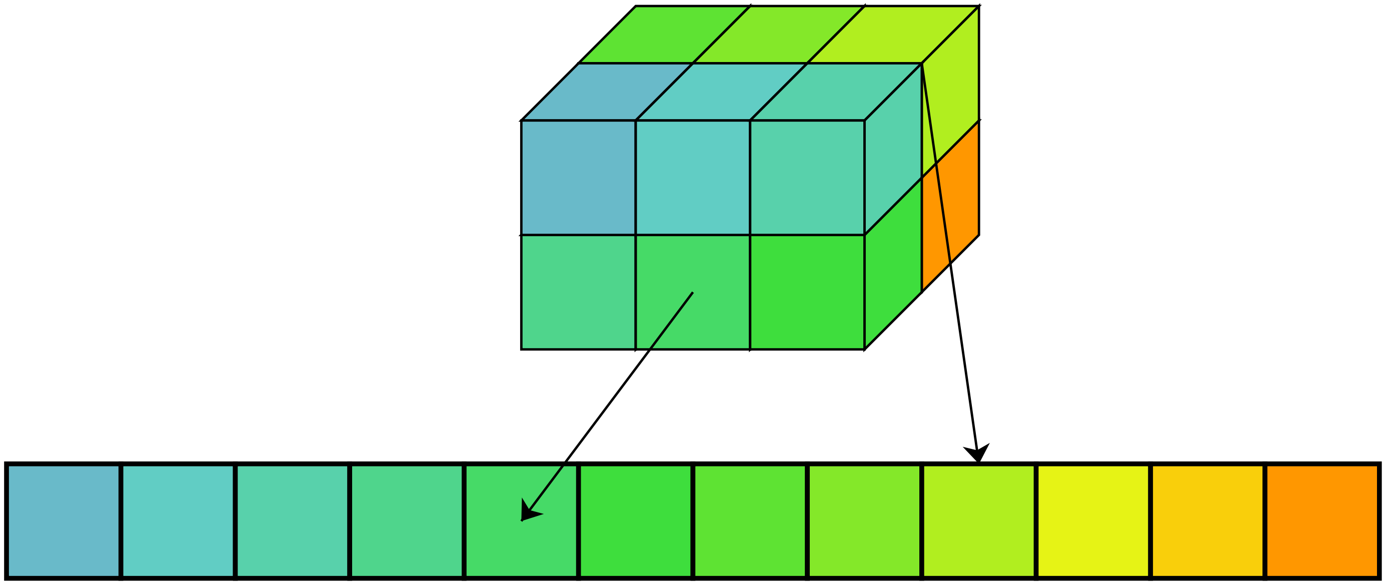
Strides



Strides



Strides



Which is best?

- Can be useful when it is contiguous
- Our Convention: Bigger strides left
- (s_1, s_2, s_3)

Operation 1: Indexing

- $v[i, j, k]$

How to find data point?

Operation 2: Movement

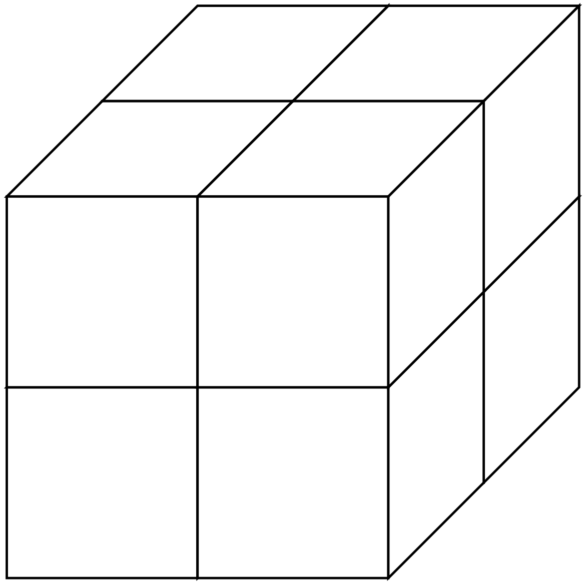
How do I move to the next in the row? Column?

Operation 3: Reverse Indexing

How do I find the index for data?

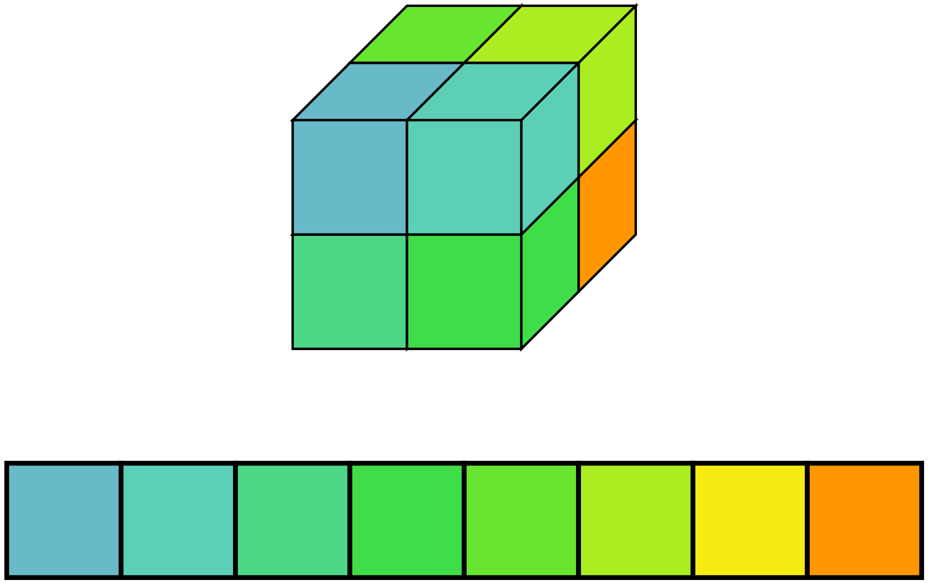
Stride Intuition

- 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
- $k = p$
- $j = (p // s_2)$
- ...

Key Operations

- Map from index to position
- Map from position to index

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

Loading [MathJax]/extensions/Safe.js