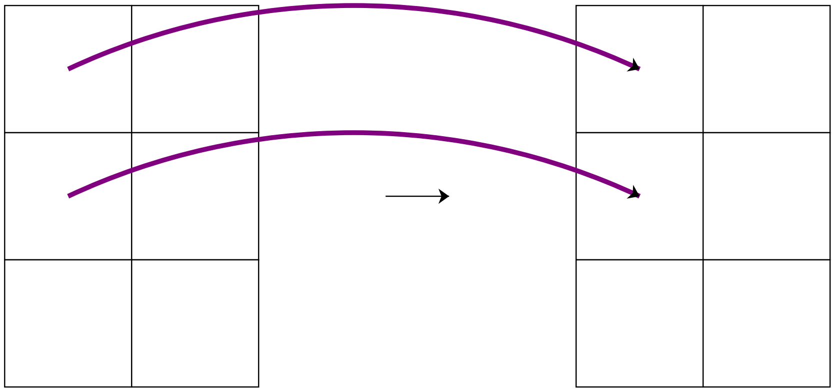
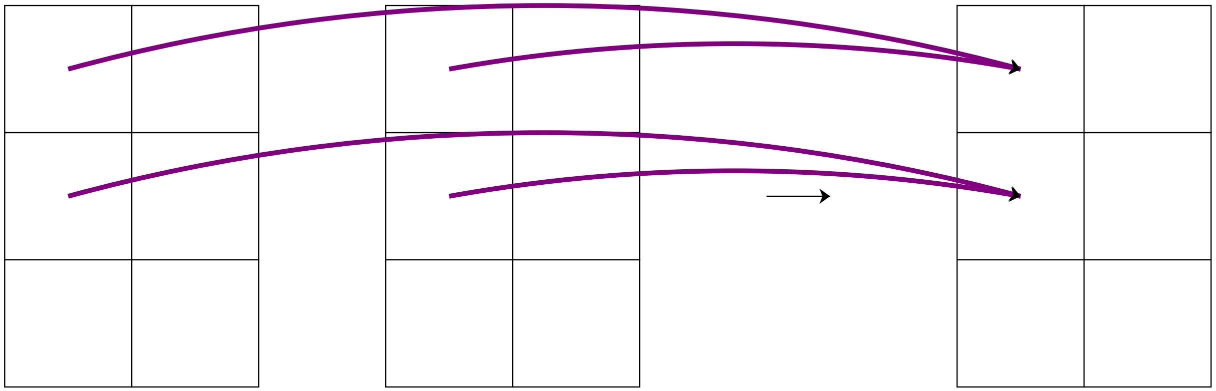
Module 2.3 - Advanced Tensors

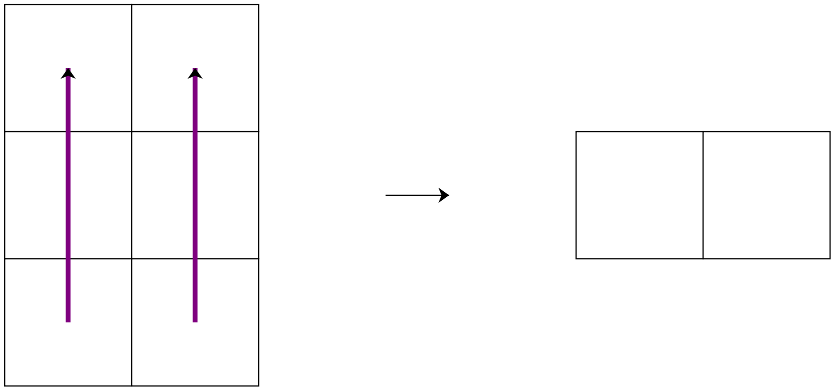
Map



Zip



Reduce



Quiz

Outline

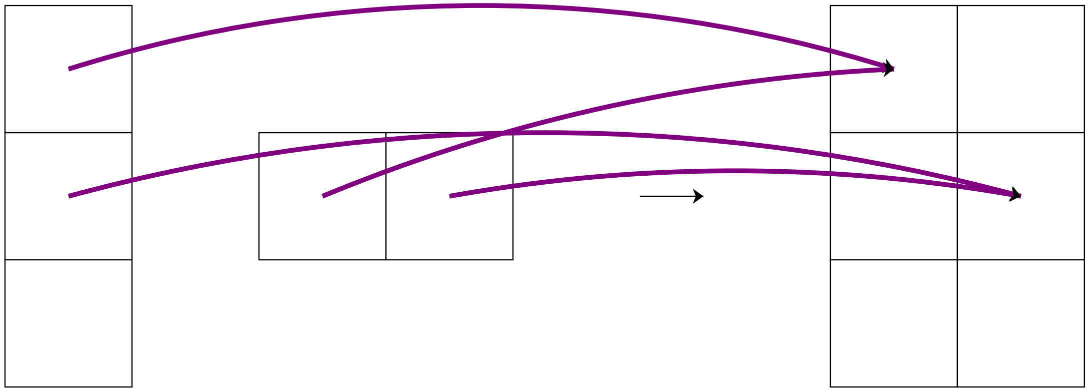
- Broadcasting
- Gradients
- Tensor Puzzles

Broadcasting

Motivation: Scalar Addition

```
vector1 + 10
```

Zip Broadcasting



Rules

- **Rule 1**: Dimension of size 1 broadcasts with anything
- **Rule 2**: Extra dimensions of 1 can be added with `view`
- **Rule 3**: Zip automatically adds starting dims of size 1

Applying the Rules

A	B	=
(3, 4, 5)	(3, 1, 5)	(3, 4, 5)
(3, 4, 1)	(3, 1, 5)	(3, 4, 5)
(3, 4, 1)	(1, 5)	(3, 4, 5)
(3, 4, 1)	(3, 5)	Fail

Broadcasting Example

```
3  4  1
   1  5
-------
3  4  5
```

View (adding dim)

```python
import torch

x = torch.tensor([1, 2, 3])
print(x.shape)
print(x.view(3, 1).shape)
print(x.view(1, 3).shape)
```

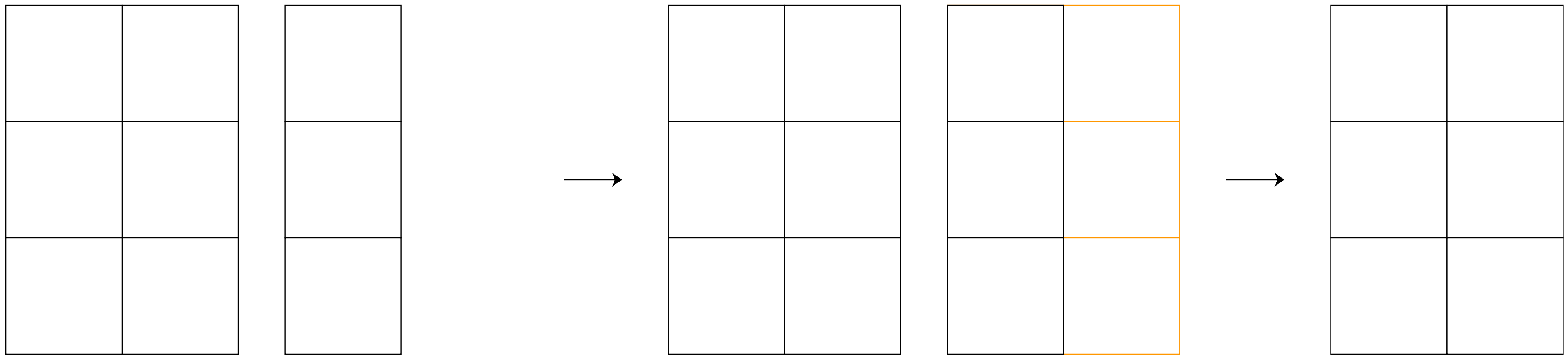
```
torch.Size([3])
torch.Size([3, 1])
torch.Size([1, 3])
```

Matrix-Vector

```python
x = minitorch.tensor([[1, 2], [3, 4], [5, 6]])
y = minitorch.tensor([[1], [3], [5]])
z = x + y
z.shape
```

```
(3, 2)
```

Matrix-Vector

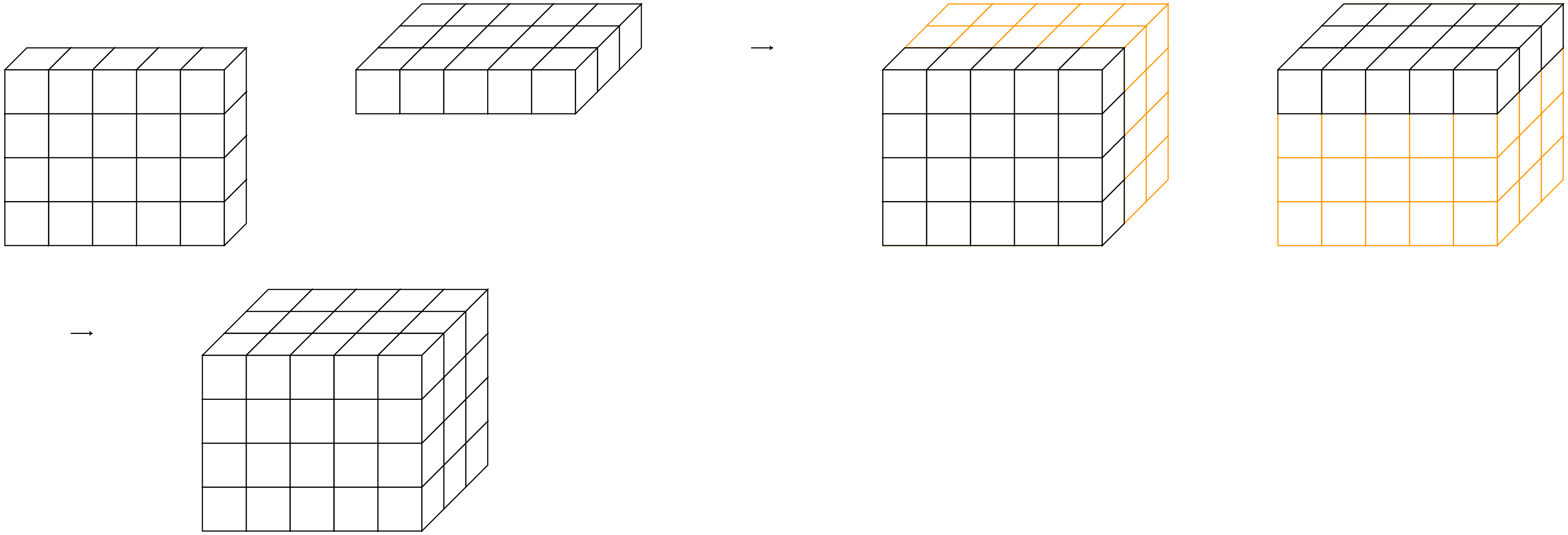


Matrix-Matrix

```
x = minitorch.zeros((4, 5))
y = minitorch.zeros((3, 1, 5))
z = x + y
z.shape
```

```
(3, 4, 5)
```

Matrix-Matrix

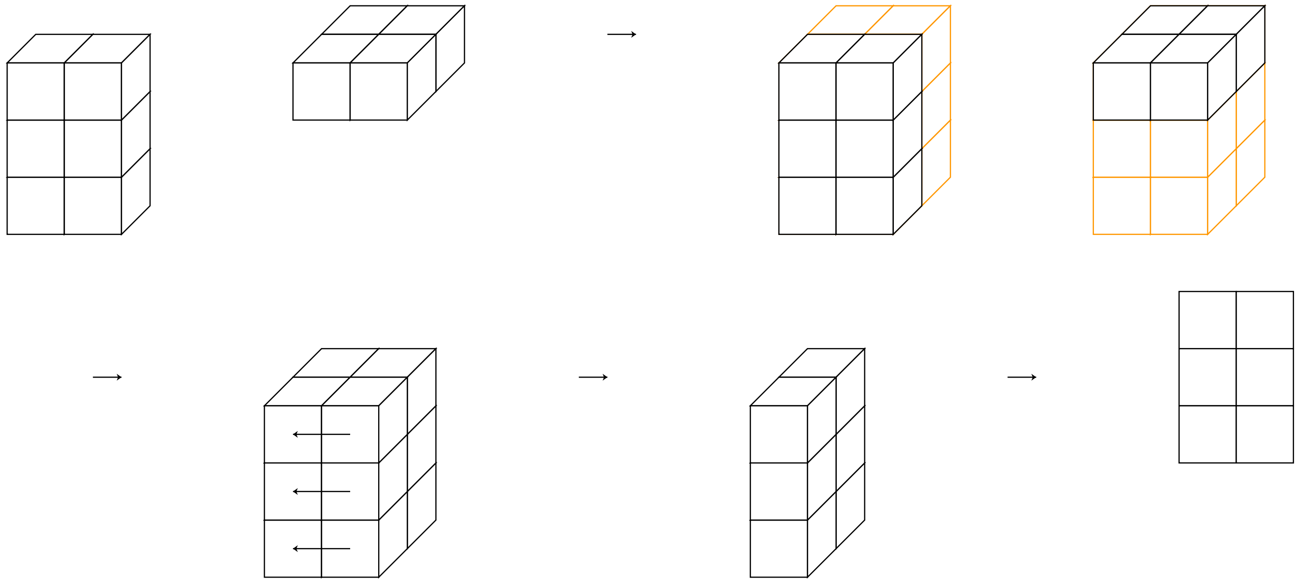


Matrix-multiplication-ish

```
x = minitorch.zeros((3, 2))
y = minitorch.zeros((2, 1, 2))
z = (x * y).sum(2)
z.shape
```

```
(2, 3, 1)
```

Matrix multiplication-ish



Implementation

Broadcast Implementation

- Never create an intermediate value.
- Implicit map between output space / input space

Broadcast Functions

- `shape_broadcast` - create the broadcast dims
- `broadcast_index` - map from broadcasted to original value

Low-level Operations

- map
- zip
- reduce

Backends

- Simple backend for debugging
- CPU implementation
- GPU implementation
- ...

Where is the backend?

- Torch: Stored on the tensor

Other Options:

- Inferred by environment
- Compiled

Low-level Operations

```
class TensorOps:
    @staticmethod
    def map(fn: Callable[[float], float]) -> Callable[[Tensor], Tensor]:
        pass

    @staticmethod
    def zip(fn: Callable[[float, float], float]) -> Callable[[Tensor, Tensor],
Tensor]:
        pass

    @staticmethod
    def reduce(
        fn: Callable[[float, float], float], start: float = 0.0
    ) -> Callable[[Tensor, int], Tensor]:
        pass
```

Constructed Operations

- Stored on tensor `tensor_op.py`

```
self.neg_map = ops.map(operators.neg)
self.sigmoid_map = ops.map(operators.sigmoid)
self.relu_map = ops.map(operators.relu)
self.log_map = ops.map(operators.log)
self.exp_map = ops.map(operators.exp)
self.id_map = ops.map(operators.id)
```

How to use

```python
t1 = minitorch.tensor([1, 2, 3])
t1.f.neg_map(t1)
```

```
[-1.00 -2.00 -3.00]
```

Implementation Tips

- Map
- Zip
- Reduce

Tensor Puzzles

Special PyTorch Syntax

(Not available in minitorch)

```python
import torch

x = torch.tensor([1, 2, 3])
print(x.shape)
print(x[None].shape)
print(x[:, None].shape)
```

```
torch.Size([3])
torch.Size([1, 3])
torch.Size([3, 1])
```

Tensor Function

```python
x = torch.arange(10)
print(x)

y = torch.arange(4)
print(y)
```

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

Tensor Function

```
x = torch.where(
    torch.tensor([True, False]),
    torch.tensor([1, 1]),
    torch.tensor([0, 0]),
)
print(x)
```

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

Tensor Puzzles

Tensor Puzzles

Q&A