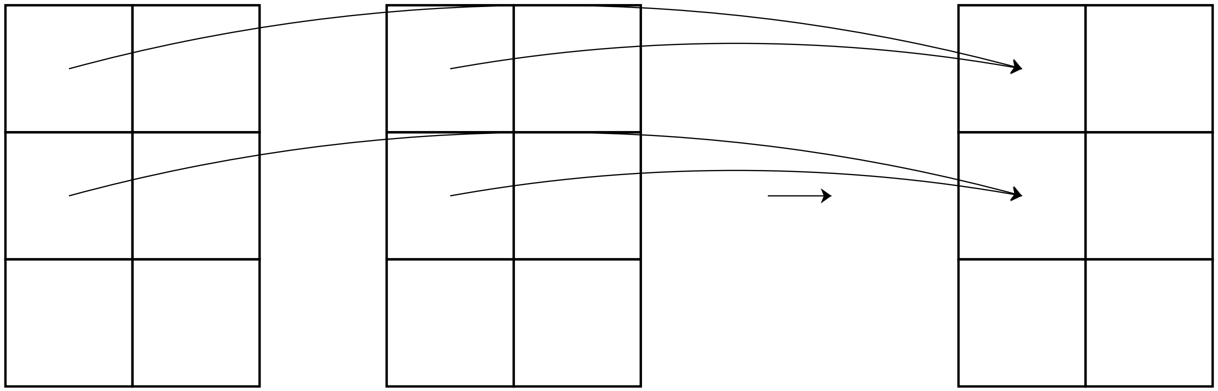
Module 2.4 - Gradients

Loading [MathJax]/extensions/Safe.js

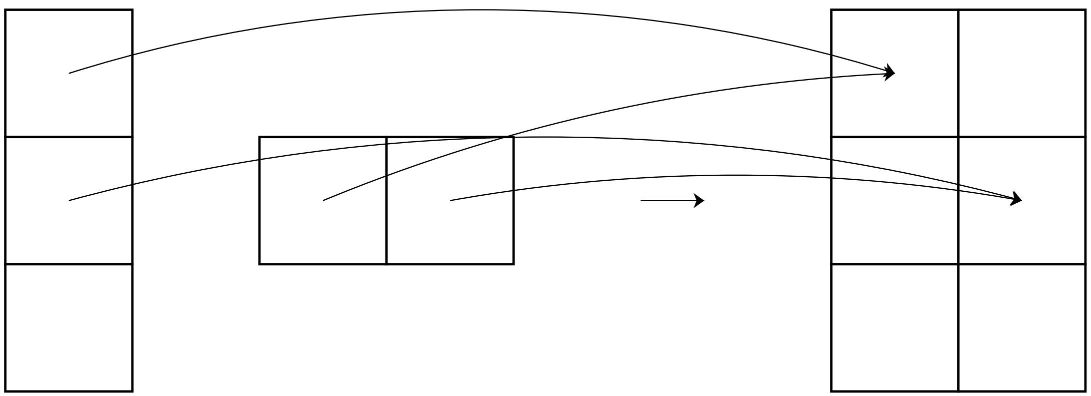
Rules

- **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

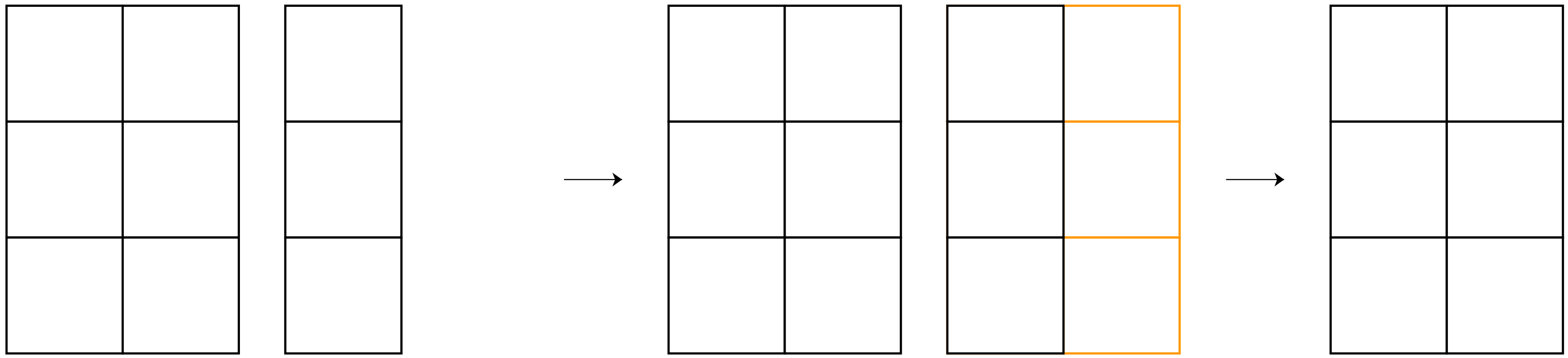
Zip



Zip Broadcasting



Matrix-Vector



Example

Quiz

Loading [MathJax]/extensions/Safe.js

Implementation

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

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

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

Implementation Tips

- Map
- Zip
- Reduce

Gradients

Loading [MathJax]/extensions/Safe.js

Derivatives

- A function with a tensor input is like multiple args
- A function with a tensor output is like multiple functions
- Backward: chain rule from each output to each input.

Terminology

- Scalar -> Tensor
- Derivative -> Gradient
- Recommendation: Reason through gradients as many derivatives

Example

What is backward?

```
x = minitorch.rand((4, 5), requires_grad=True)
y = minitorch.rand((4, 5), requires_grad=True)
z = x * y
z.sum().backward()
```

Notation: Gradient

Function from tensor to a scalar

$$f([x_1, x_2, \ldots, x_N])$$

Gradient

$$f'_{x_1}([x_1, x_2, \ldots, x_N])$$
$$f'_{x_2}([x_1, x_2, \ldots, x_N])$$
$$\ldots$$
$$f'_{x_N}([x_1, x_2, \ldots, x_N])$$

Each is a standard derivative

Gradient

$$
\begin{aligned}
[f'_{x_1}([x_1, x_2, \ldots, x_N]),\\
f'_{x_2}([x_1, x_2, \ldots, x_N]),\\
\ldots\\
f'_{x_N}([x_1, x_2, \ldots, x_N])]
\end{aligned}
$$

Tensor of derivatives.

Function to Tensor

Function to a tensor

$$G(x)$$

Function to Tensor

Think of it as many functions

$$g^1(x), g^2(x), \ldots, g^N(x)$$

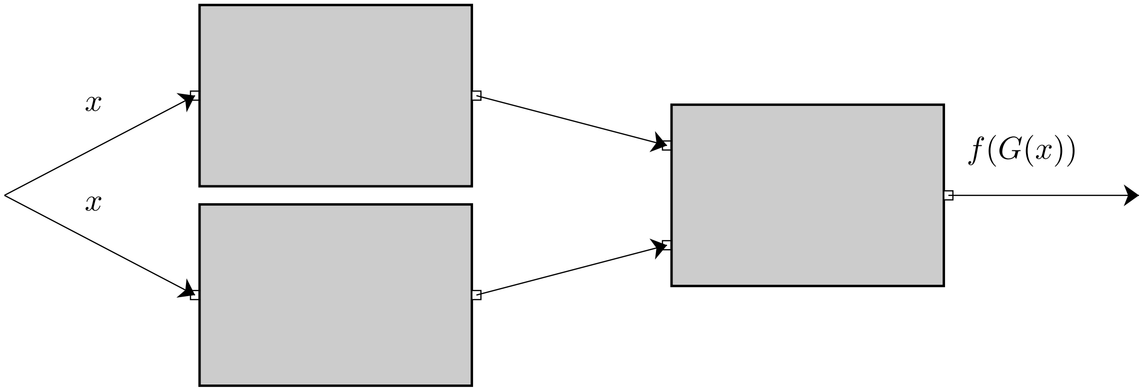
Function to Tensor

Think of it as many functions

$$G(x) = [g^1(x), g^2(x), \ldots, g^N(x)]$$

Example: Chain Rule For Gradients

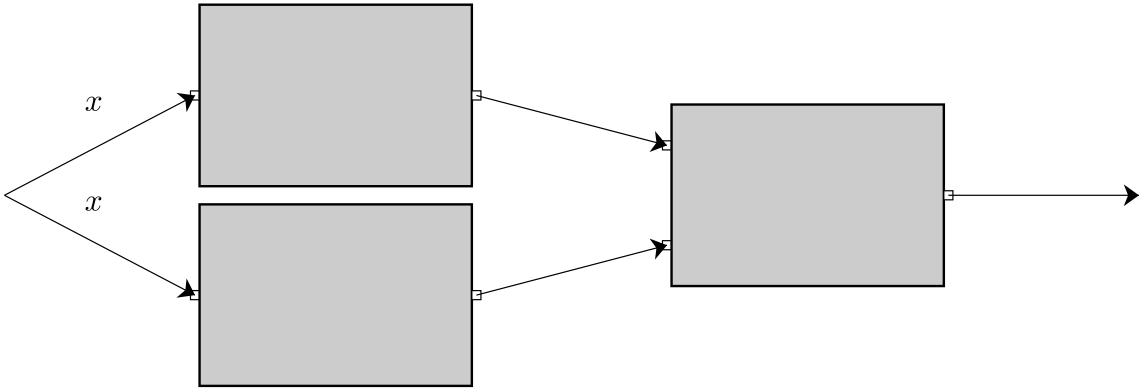
- $G(x) = [g^1(x), g^2(x)]$ - scalar to tensor
- $f(x)$ - tensor to scalar



Review: Chain Rule

$f(G(x))$

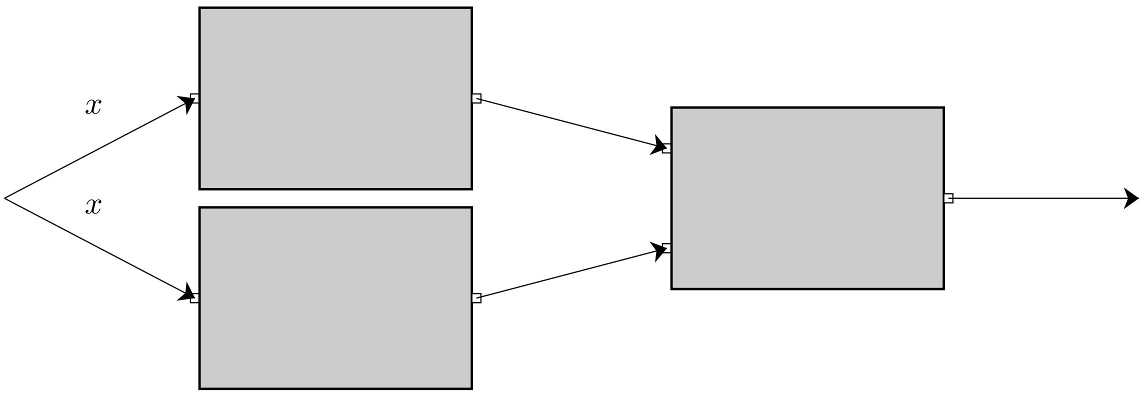
- $z_1 = g^1(x)$, $z_2 = g^2(x)$
- $d_1 = f'_{z_1}(z_1, z_2), d_2 = f'_{z_2}(z_1, z_2)$
- $f'_x(G(x)) = d_1 g'^1_x(x) + d_2 g'^2_x(x)$



Review: Chain Rule

$$f(G(x))$$

- $z_1 = g^1(x), z_2 = g^2(x), \ldots$
- $d_1 = f'_{z_1}(z), d_2 = f'_{z_2}(z), \ldots$
- $f'_{x}(G(x)) = \sum_i d*i g^{'i}*{x}(x)



Tensor-to-Tensor

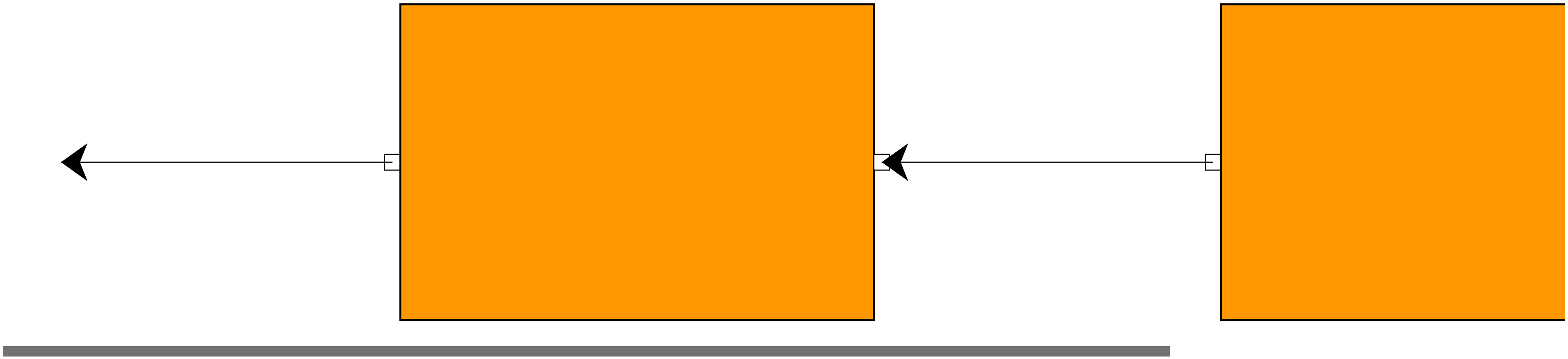
$$G([x_1, \ldots, x_N]) = [G^1([x_1, \ldots, x_N]), \ldots]$$

Chain Rule For Gradients

$f(G(x))$

- $z_1 = G^1(x), z_2 = G^2(x), \ldots$
- $d_1 = f'_{z_1}(z), d_2 = f'_{z_2}(z), \ldots$
- $f'_{x_j}(G(x)) = \sum_i d_i G'^i_{x_j}(x)$

Chain Rule For Gradients



Avoiding Gradient Math

- All of this is just notation for scalars
- Can often reason about it with scalars directly

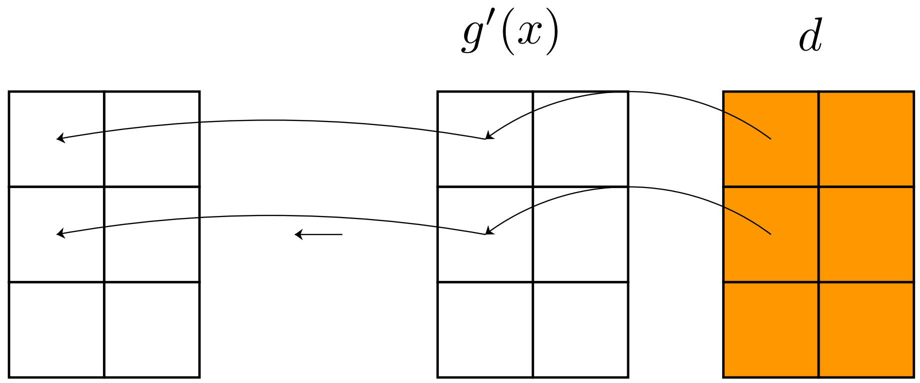
Special Function: Map

$$G'^{i}_{x_j}([x_1, \ldots, x_N])\ ?$$

Special Function: Map

- $G'^{i}_{x_j}(x) = 0$ if $i \neq j$
- $f'_{x_j}(G(x)) = d_i g'^{j}_{x_j}(x)$

Map Gradient



Loading [MathJax]/extensions/Safe.js

Example: Negation

```python
class Neg(minitorch.ScalarFunction):
    @staticmethod
    def forward(ctx, a: float) -> float:
        return -a

    @staticmethod
    def backward(ctx, d: float) -> float:
        return -d
```

Example: Tensor Negation

```
class Neg(minitorch.Function):
    @staticmethod
    def forward(ctx, t1: Tensor) -> Tensor:
        return t1.f.neg_map(t1)

    @staticmethod
    def backward(ctx, d: Tensor) -> Tensor:
        return d.f.neg_map(d)
```

Example: Inv

```
class Inv(minitorch.Function):
    @staticmethod
    def forward(ctx, t1: Tensor) -> Tensor:
        ctx.save_for_backward(t1)
        return t1.f.inv_map(t1)

    @staticmethod
    def backward(ctx, d: Tensor) -> Tensor:
        (t1,) = ctx.saved_values
        return d.f.inv_back_zip(t1, d)
```

Special Function: Zip

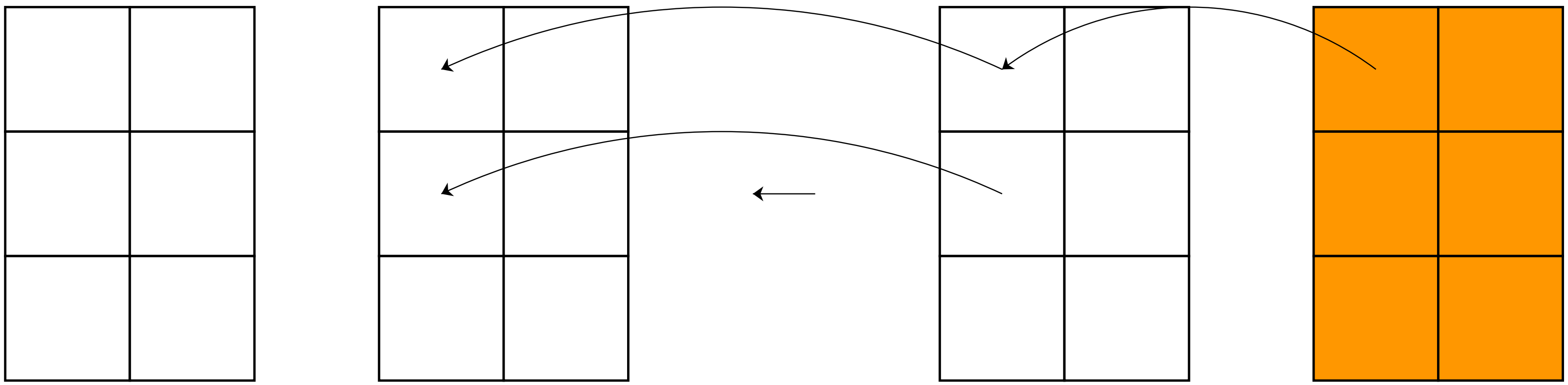
$G'^{i}_{x_j}(x, y)$?

Loading [MathJax]/extensions/Safe.js

Special Function: Map

- $G'^{i}_{x_j}(x) = 0$ if $i \neq j$
- $f'_{x_j}(G(x)) = d_i g'^{j}_{x_j}(x, y)$

Zip Gradient



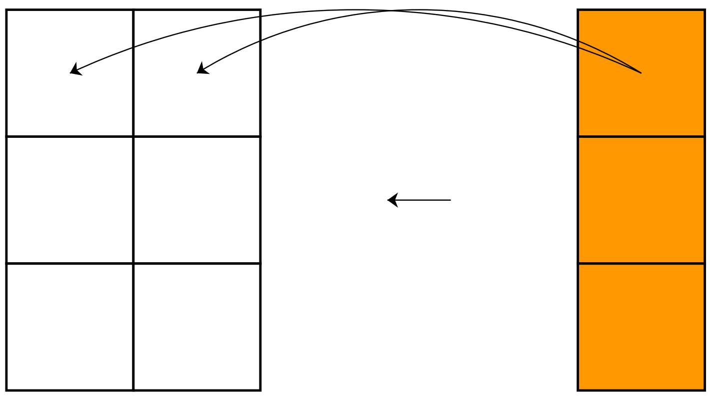
Loading [MathJax]/extensions/Safe.js

Example: Add

```
class Add(minitorch.Function):
    @staticmethod
    def forward(ctx, t1: Tensor, t2: Tensor) -> Tensor:
        return t1.f.add_zip(t1, t2)

    @staticmethod
    def backward(ctx, grad_output: Tensor) -> Tuple[Tensor, Tensor]:
        return grad_output, grad_output
```

Reduce Gradient



Example: Sum

```python
class Sum(minitorch.Function):
    @staticmethod
    def forward(ctx, a: Tensor, dim: Tensor) -> Tensor:
        ctx.save_for_backward(a.shape, dim)
        return a.f.add_reduce(a, int(dim.item()))

    @staticmethod
    def backward(ctx, grad_output: Tensor) -> Tuple[Tensor, float]:
        a_shape, dim = ctx.saved_values
        return grad_output, 0.0
```

Q&A