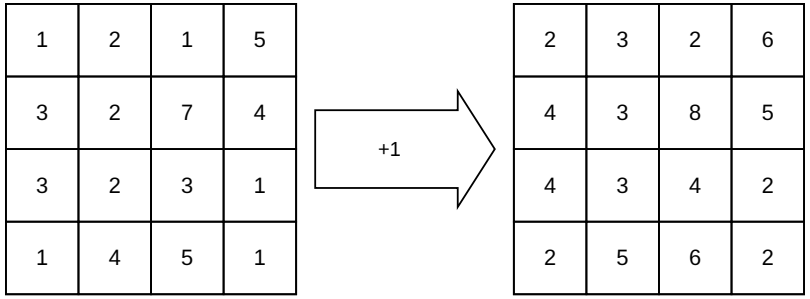
Regular Data Parallelism

Regular Arrays

Multi-dimensional flat arrays (not ragged).

Examples: raster graphic pictures, medical imaging volumes, neural network weights, ..., vectors, matrices, tensors, ...

Single Instruction Multiple Data (SIMD) parallelism.



When to use it?

Whenever applicable! This is what works!

This is the simplest yet an immensely important form of parallelism!

"Embarrissingly Parallel"

Examples: physics simulation, scientific computation, machine learning, ...

"Number Crunching"

PyTorch

https://pytorch.org/

> [...] and does so while maintaining performance comparable to the fastest current libraries for deep learning. This combination has turned out to be very popular in the research community with, for instance, 296 ICLR 2019 submissions mentioning PyTorch.

Adam Paszke, Sam Gross, Francisco Massa, et al. 2019. PyTorch: an imperative style, high-performance deep learning library.

Python

Interpreted, untyped programming language.

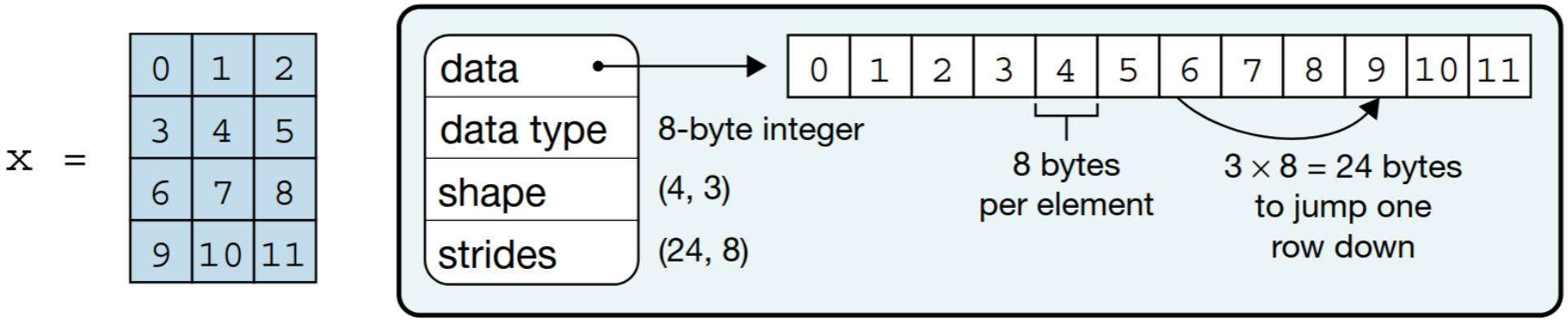
```
factorial = 1

for i in range(2, n + 1):
    factorial *= i

print(factorial)
```

Created by Guido van Rossum in 1991.

Python Package Index: rich ecosystem of useful packages.

torch.Tensor

https://docs.pytorch.org/docs/stable/tensors.html



Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. Array programming with NumPy.

Element-wise Operations

Assuming `a` and `b` are vectors, this exploits parallelism:

```
c = a + b
```

It is that simple!

demo/basics.py

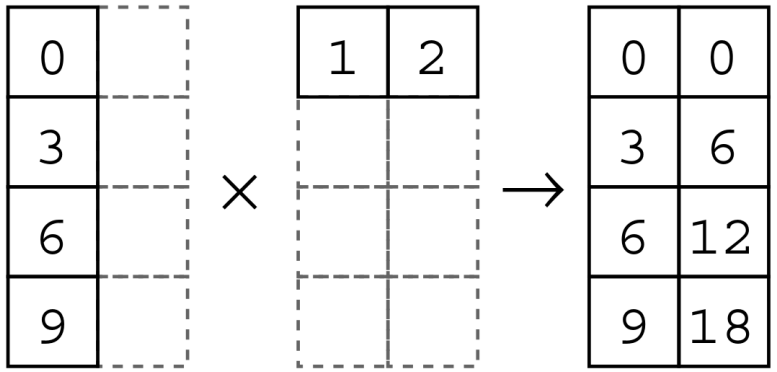
Broadcasting

What does the following mean when `a` is a vector?

```
2 * a
```

Scalar multiplication!

Not limited to scalars:



Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. Array programming with NumPy.

Broadcasting in PyTorch

https://docs.pytorch.org/docs/stable/notes/broadcasting.html

Two tensors are “broadcastable” if the following rules hold:

- Each tensor has at least one dimension.
- When iterating over the dimension sizes, starting at the trailing dimension, the dimension sizes must either be equal, one of them is 1, or one of them does not exist.

If two tensors x, y are “broadcastable”, the resulting tensor size is calculated as follows:

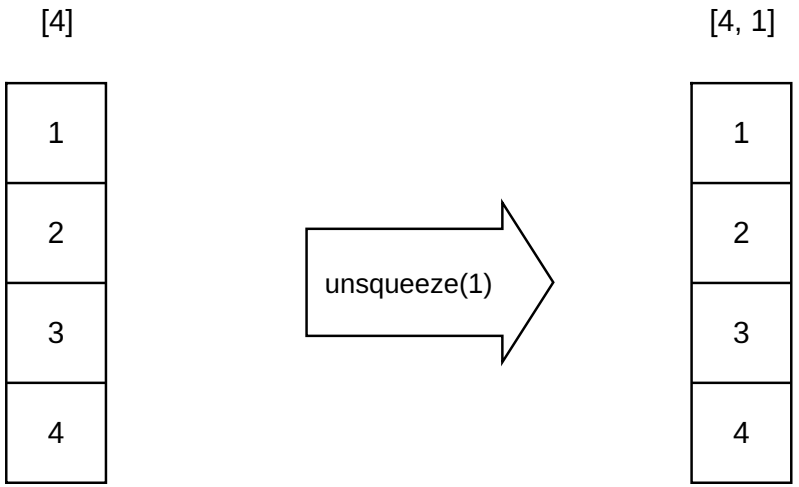
- If the number of dimensions of x and y are not equal, prepend 1 to the dimensions of the tensor with fewer dimensions to make them equal length.
- Then, for each dimension size, the resulting dimension size is the max of the sizes of x and y along that dimension.

torch.unsqueeze

```
torch.unsqueeze(input, dim) -> Tensor
```

Returns a new tensor with a dimension of size one inserted at the specified position.

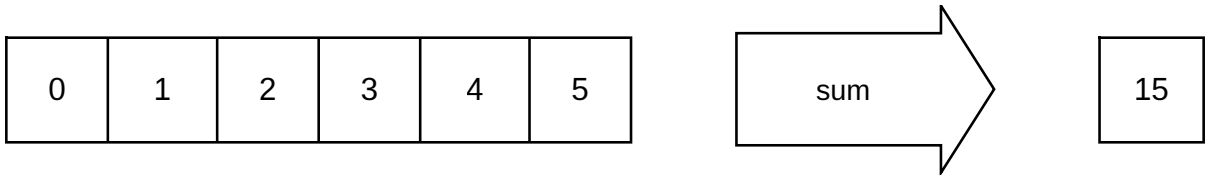
Data stays the same, shape is extended.



demo/broadcasting.py

Reduction

Aggregation functions `sum`, `max`, etc.



Associative operation: `(a + b) + c = a + (b + c)`

```
((((0 + 1) + 2) + 3) + ...)
```

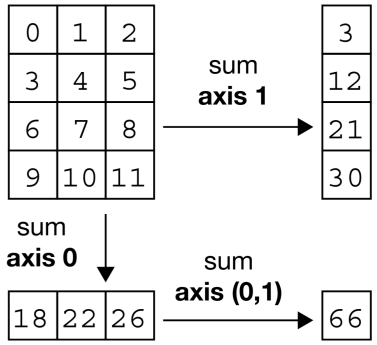
Parallel tree-shaped reduction:

```
(0 + 1) + (2 + 3) + ...
```

Reduction in PyTorch

```
x = torch.tensor([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])

print(x.sum(dim=1))
print(x.sum(dim=0))
print(x.sum(dim=(0, 1)))
```

Parallel along the other axis:



Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. Array programming with NumPy.

demo/reductions.py

Example: Softmax

https://en.wikipedia.org/wiki/Softmax_function

$$\sigma(\mathbf{z})_i = \frac{e^{\beta(z_i - m)}}{\sum_{j=1}^{K} e^{\beta(z_j - m)}}$$

where $m = \max_i z_i$ is the largest factor involved.

```
def softmax(x: torch.Tensor) -> torch.Tensor:
    y = torch.exp(x - x.max())
    return y / y.sum()
```

demo/softmax.py

Vectorization

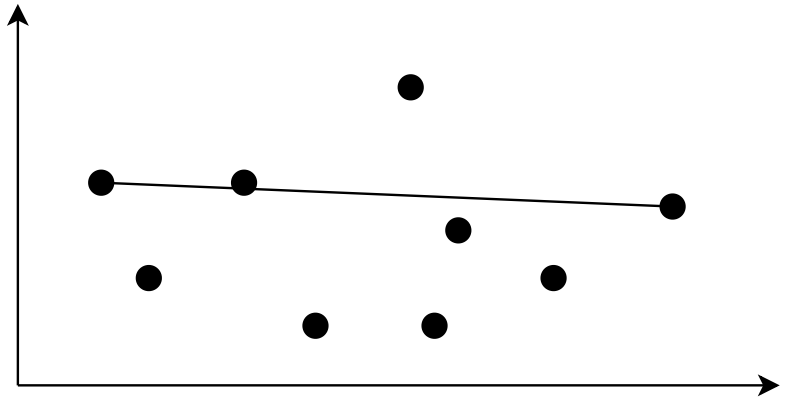
Sequential loops are slow, parallel loops are fast.

Rewrite programs to use tensor operations instead of loops!

Beware of data dependencies between loop iterations.

Vectorization Example

Maximum distance between points in 2D space.



```
for i in range(size):
    for j in range(size):
        distances[i, j] = torch.norm(points[i] - points[j])

print(distances.max())
```

demo/distances.py

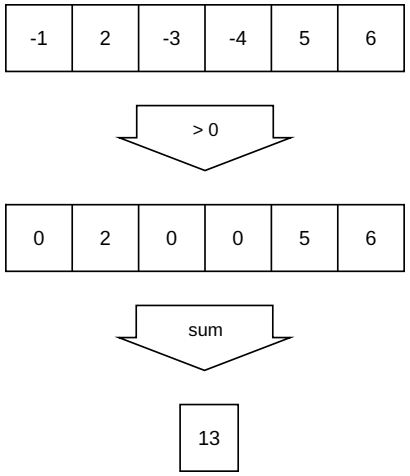
Conditions and Masking

There is no control flow in data parallelism.

We must emulate control flow with conditional data flow.

This might perform unnecessary work.

Example: sum of positive numbers.



Conditions and Masking in PyTorch

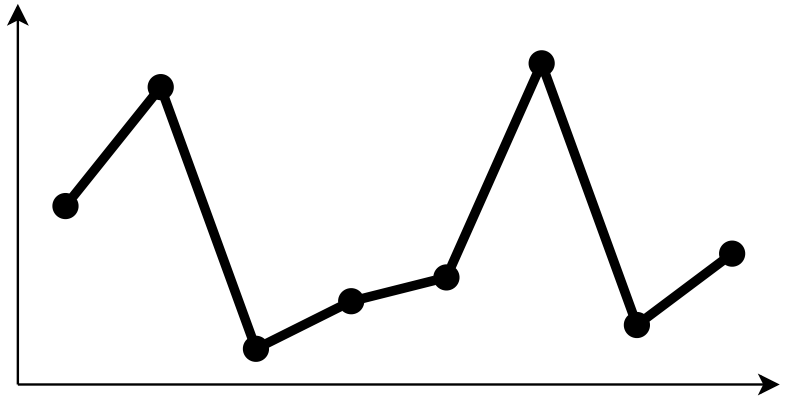
```
x = torch.tensor([-1, 2, -3, -4, 5, 6])
mask = x > 0
filtered = torch.where(mask, x, torch.tensor(0))
result = filtered.sum()
```

demo/conditions.py

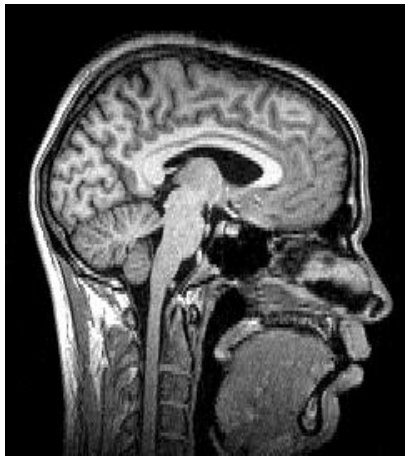
Convolution

Locality between indexes in domain.

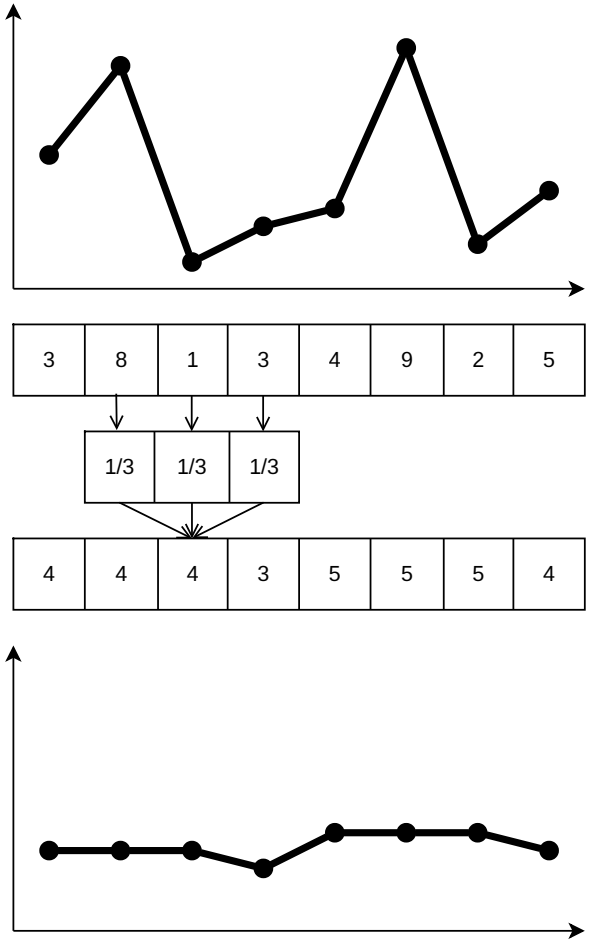
Examples: Sound and Image Processing, Tomograms, Fluid simulations, ...



3	8	1	3	4	9	2	5



Convolution in PyTorch



Sequential

```
input = [3, 8, 1, 3, 4, 9, 2, 5]
result = [0] * len(input)

for i in range(1, len(input) - 1):
    result[i] = (1/3) * input[i-1] + (1/3) * input[i] + (1/3) * input[i+1]
```

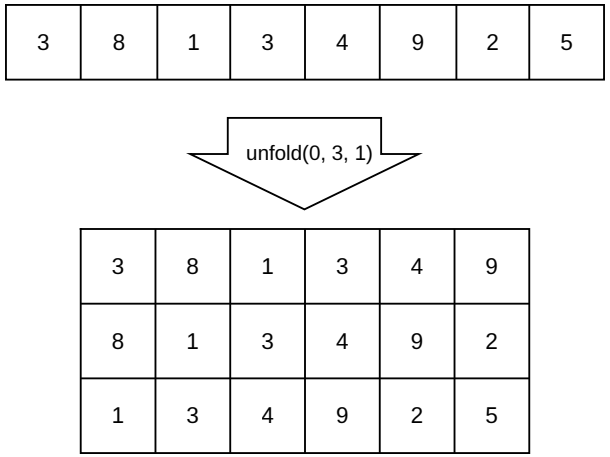
Parallel

```
input = torch.tensor([3, 8, 1, 3, 4, 9, 2, 5])
kernel = torch.tensor([1/3, 1/3, 1/3])

windows = input.unfold(0, 3, 1)
result = (kernel * windows).sum(dim=1)
```

torch.unfold

```
Tensor.unfold(dimension, size, step) -> Tensor
```

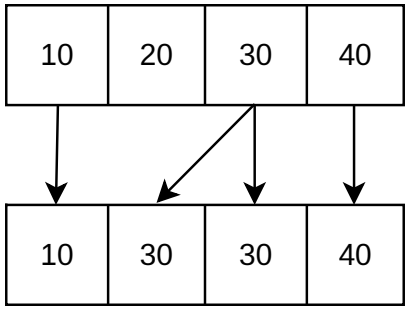
Returns a view of the original tensor which contains all slices of size size from self tensor in the dimension dimension. Step between two slices is given by step.



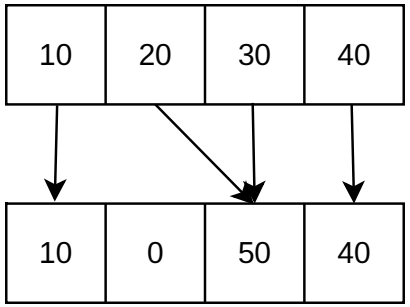
demo/convolution.py

Irregular data parallelism

Gather



Scatter



Irregular data parallelism in PyTorch

Gather

```
x = torch.tensor([10, 20, 30, 40])
i = torch.tensor([0, 2, 2, 3])
r = x[i]  # tensor([10, 30, 30, 40])
```

Scatter

```
r = torch.zeros(4)
x = torch.tensor([10, 20, 30, 40])
i = torch.tensor([0, 2, 2, 3])
r = r.scatter_add(0, i, x)  # tensor([10, 0, 50, 40])
```

demo/gather.py

demo/scatter.py

Summary and Outlook

Regular data parallelism is great.

Next week: sparse and nested tensors.