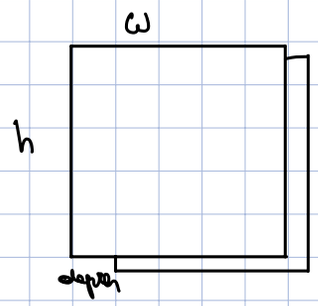
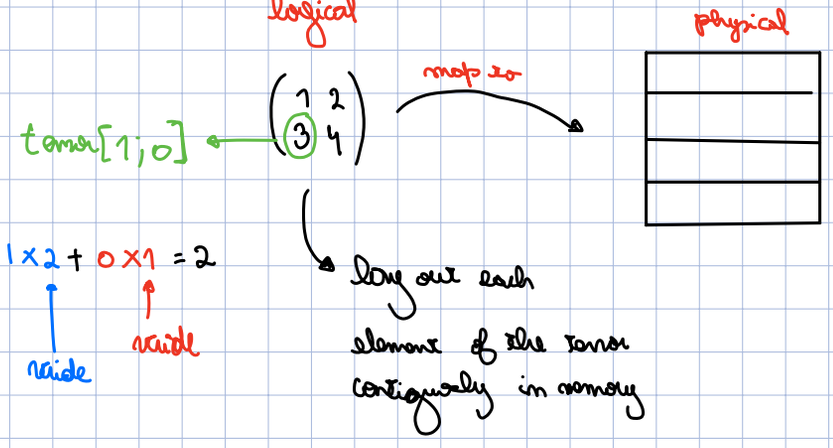
tensor

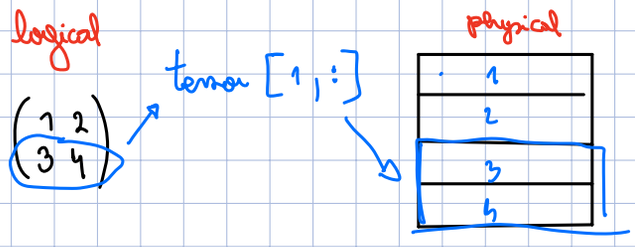


stride (h*w; w; 1)

logical

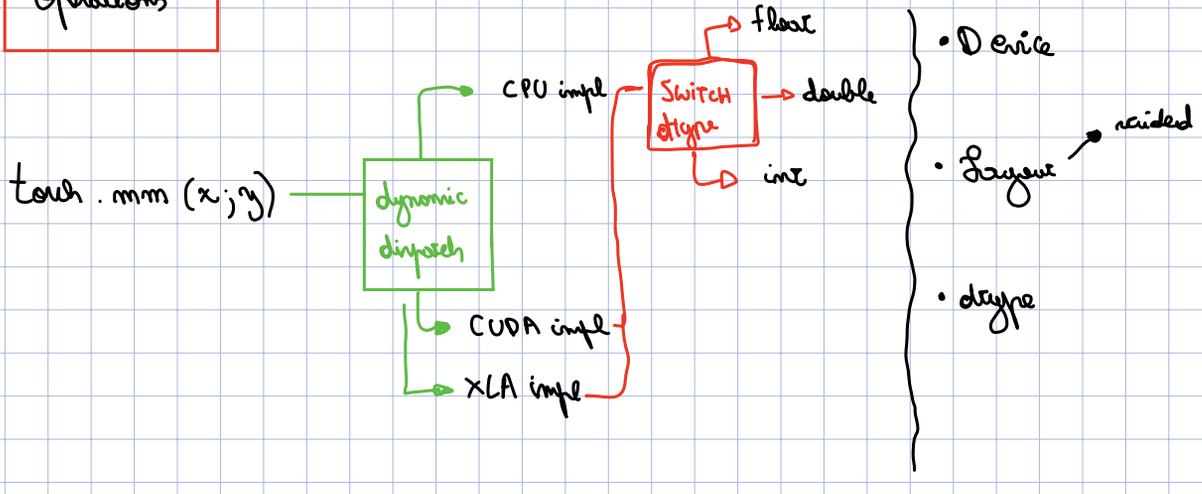


other example



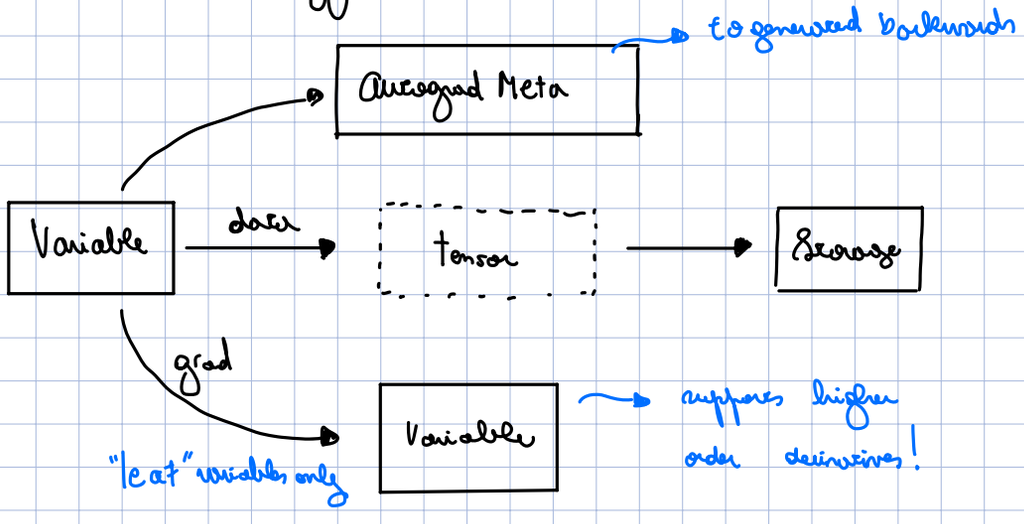
DOES NOT CREATE A NEW TENSOR

Operations



Autograd (automatic differentiation)

automatic differentiation on tensors

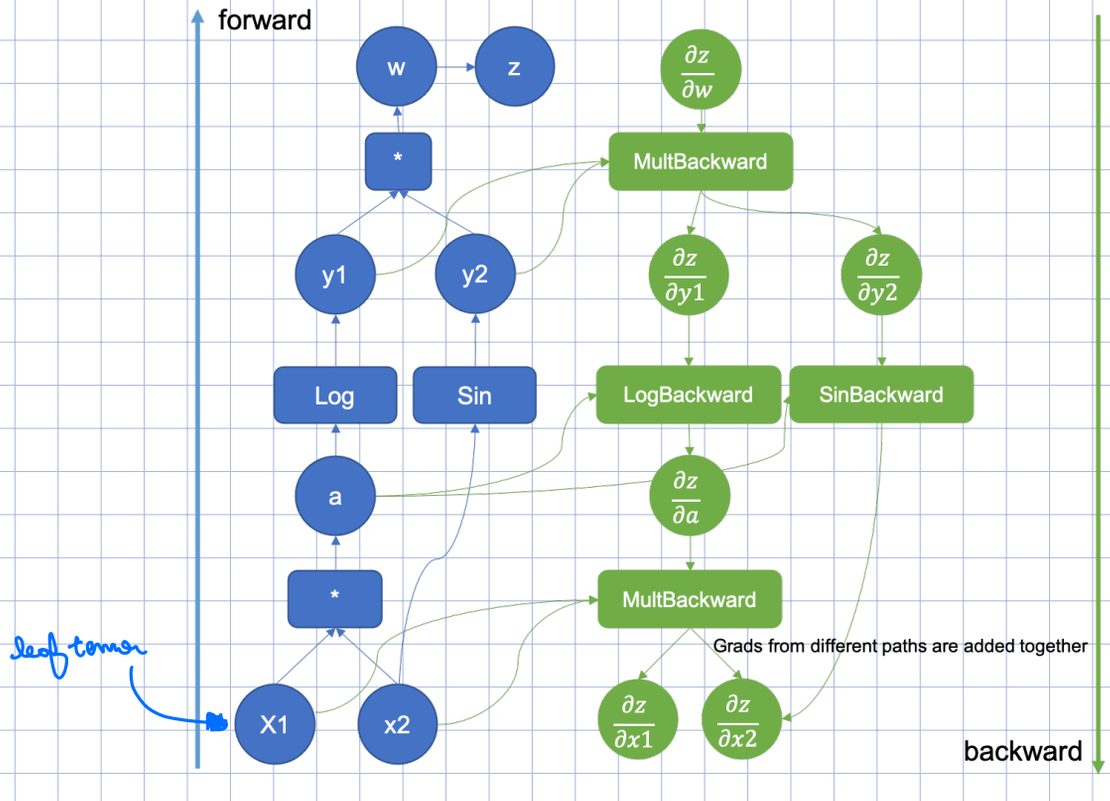


Computational graphs

↳ torch.autograd.function → class to write your own differentiable functions

↳ for jacobian vector product, hessian

graph creation



if requires_grad is set then an AutogradMeta will be allocated

- Computed grad accum
- grad_fn → differentiable function

```python
import torch

# Create input tensors
x = torch.tensor([2.0], requires_grad=True)
y = torch.tensor([3.0], requires_grad=True)

# Perform addition
z = x + y  # z is the result of addition
print("z.grad_fn:", z.grad_fn)  # Output: <AddBackward0>

# Perform multiplication
w = z * 2  # w is the result of multiplication
print("w.grad_fn:", w.grad_fn)  # Output: <MulBackward0>
```

grad_fn → represents a node in comp graph

!!! Uses chain Rule

- input tensors that are connected
- compute gradients (partial derivative)

the grad_fn could hold them (tensors) using the set_next_edge

↓

copies tensors to AutogradMeta

Differentiation in Autograd

↳ requires_grad = true → signals that every operation on them should be tracked

$$Q = 3 \cdot a^3 - b^2$$

↑ assume it to be the error in a NN

$$\frac{\partial Q}{\partial a} = 9a^2$$

$$\frac{\partial Q}{\partial b} = -2b$$

→ calculates these grads and store them in .grad

$$a = (2; 3)$$

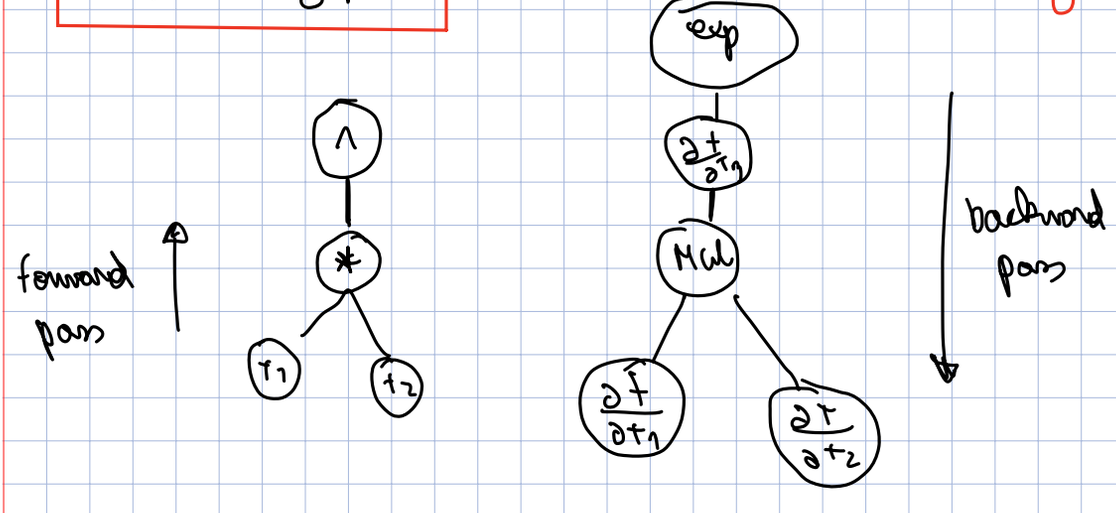
$$b = (6; 4)$$

$$Q(a,b) = 3a^3 - b^2$$

$$Q : \mathbb{R}^2 \to \mathbb{R} \quad \text{(should always be on scalar func)}$$

$$\vec{\nabla} Q = \left(\frac{\partial Q}{\partial a} ; \frac{\partial Q}{\partial b} \right)$$

Computational graph → stored in a directed acyclic graph (DAG)



When you done want to compute gradients of certain parameters you freeze them

Batch normalization → couples example (avoid)

Linearity → Normalization → Non-linearity