Pytorch

torch → core module

torch.autograd → automatic differentiation

torch.nn → provides a neural network library
- activation
- loss fn
- utilities to build nn

torch.optim → optimizer algorithms
- SGD
- Adam
- RMSprop.

torch.utils.data → Dataset & Dataset Loader
classes

torch.jit → torch script

Pytorch Domain Libraries

torchvision, torchtext, torchaudio → speech recognition

torchvision → Computer vision

torchtext → NLP

Tensors

Tensor is a specialized multi-dimensional array designed for mathematical & computational efficiency

① Scalar

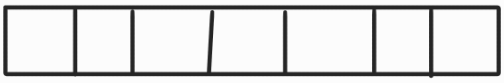
Rank 0 

(scalar)

eg 5 , 3.14

② Vector

Rank 1:



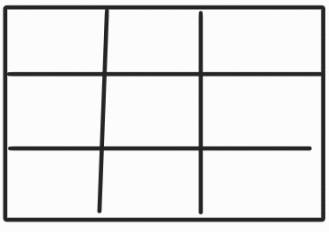
vector

eg: [0.12, -0.84, 0.33]

word embedding

③ Matrices — 2D tensor

represents tabular or grid like data.

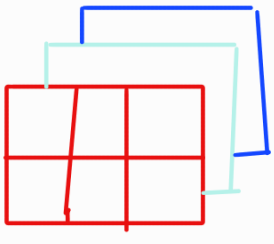


[[0, 255, 128],
[34, 90, 180]]

④ 3D Tensor.

Adds 3rd dimention

eg: Colored img



⑤ 4D Tensor

eg Batch of RGB example

⑥ 5D Tensor

eg Video data.

Why are tensor used in Deep Learning

1. Mathematical operation (+, x, dot product etc)
2. Representation of real world. → audio, text, img, video
3. efficient computation → parallel processing

Where are tensor used.

1. Data Storage → (img, text etc)
2. wt & Biases
3. Matrix operations. → activation (wx + b)
4. Training process
   - ↳ forward pass
   - gradient calculation

Pytorch Autograd

$y = x^2$ → program → (x) → $x \rightarrow \frac{dy}{dx}$

$\frac{dy}{dx} = 2x$ code

```
diff_sq(x^2)
return 2x
```

torch.autograd → automatic differentiation engine

Traning in NN happens in 2 steps

1. Forword Propagation
2. Backword Propagation

autograd → traces your computation dynamically at the runtime.

* Each `grad_fn` stored with tensor allows you to walk the computation all the way back to its input with its `next_function` property.

$y = x^2$

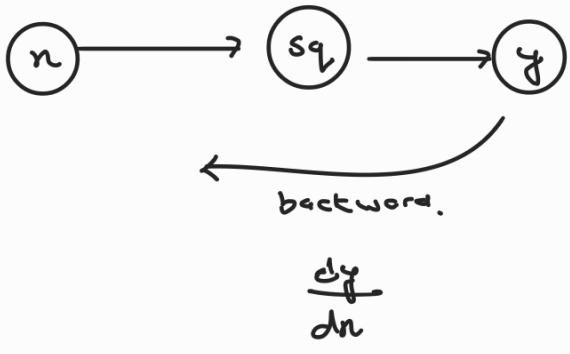
$z = \sin(y)$

$\frac{dz}{dy} = \cos y$ $\qquad$ $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

$= 2x\cos(y)$

in NN → Nested function calculating derivative manually is almost impossible

$y = x^2$



$\frac{dy}{dx}$

```
x = torch.tensor(3.0, require-grad =Ture)
y =x**2
y.backward()
x.grad

= tensor(6.)
```

for enabling gradient calculation

$x = 3$

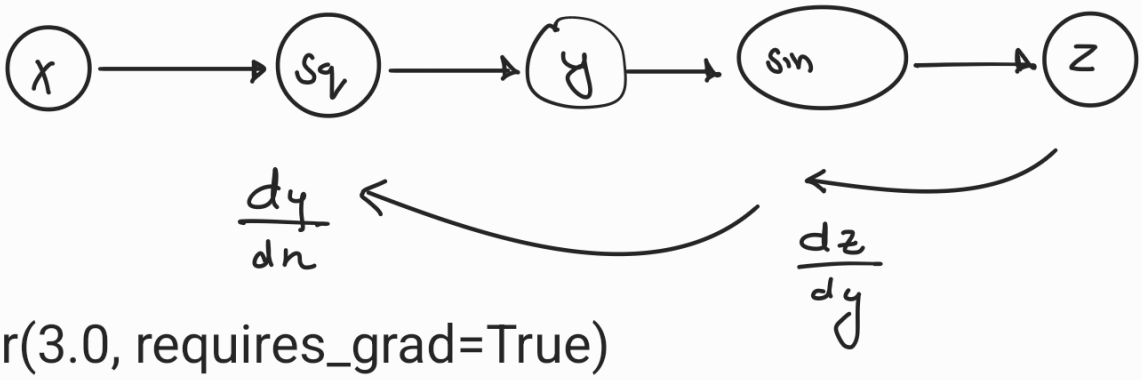
$y = x^2$

$\frac{dy}{dx} = 2x$

$2 \times 3 = 6$

②

$y = x^2$, $z = \sin(y)$




```
x=torch.tensor(3.0, requires_grad=True)
y=x**2
z=torch.sin(y)
z.backward()
x.grad
= tensor(-5.4668)
```

$y = x^2$

$z = \sin(y)$

$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

$= \cos(y) \cdot \frac{dy}{dx}$

$= \cos(x^2) \cdot 2x$

$= 2x \cos(x^2)$

Conceptually, autograd keeps a record of data (tensors) and all executed operations (along with the resulting new tensors) in a DAG (Directed Acyclic graph) consisting of Functions of Object.

In forwardpass, autograd

- run the requested operation to compute a resulting tensor

- maintain the operation's gradient function in the DAG

The backward pass kicks off when .backward() is called on the DAG root. autograd.

- computes the gradient from each .grad_fn.
- accumulates them in the respective tensor's .grad attribute.

- using chain rule propogates all the way to the leaf tensors.

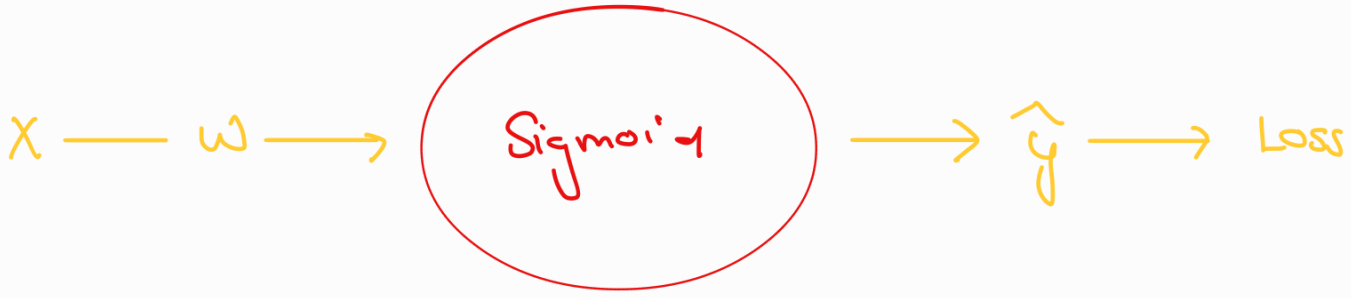
$$L = -[y \cdot \ln(\hat{y}) + (1-y)\ln(1-\hat{y})]$$

$$\frac{\partial L}{\partial \hat{y}} = -\left(y \cdot \frac{1}{\hat{y}} + \frac{(1-y)}{1-\hat{y}}\right)$$

$$= -\left(\frac{y(1-\hat{y}) - \hat{y}(1-y)}{\hat{y}(1-\hat{y})}\right)$$

$$= -\left(\frac{y - y\hat{y} - \hat{y} + y\hat{y}}{\hat{y}(1-\hat{y})}\right)$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$



① Linear Transformation

$$z = w.x + b$$

② Activation (Sigmoid function)

$$y_{pred} = \sigma(z) = \frac{1}{1+e^{-z}}$$

③ Loss function (Binary Cross Entropy Loss)

$$L = [y_{target} \cdot \ln(y_{pred}) + (1 - y_{target}) \cdot \ln(1 - y_{pred})]$$

where $y_{target} = y$

$y_{pred} = \hat{y}$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y_{pred}} \frac{\partial y_{pred}}{\partial z} \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y_{pred}} \frac{\partial y_{pred}}{\partial z} \frac{\partial z}{\partial b}$$

Sigmoid $\quad \sigma(z) = \frac{1}{1+e^{-z}}$

$$= (1+e^{-z})^{-1}$$

$$\frac{\partial \sigma}{\partial z} = -1 (1+e^{-z})^{-2} \cdot (-e^{-z})$$

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

Noting that

$$\sigma(z) = \frac{1}{1+e^{-z}} \implies 1 - \sigma(z) = \frac{e^{-z}}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z)\,(1 - \sigma(z)).$$

Thus

$$\frac{\partial}{\partial z}\sigma(z) = \sigma(z)\,(1-\sigma(z))$$

since $\hat{y} = y_{pred} = \sigma(z) = \frac{1}{1+e^{-z}}$

$$\frac{\partial y_{pred}}{\partial z} = \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

$$\frac{\partial z}{\partial w} = x$$

$$\frac{\partial z}{\partial b} = 1$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w} = \frac{(\hat{y}-y)}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) \cdot x = (\hat{y}-y)\cdot x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial b} \cdot \frac{(\hat{y}-y)}{\hat{y}(1-\hat{y})}\,\hat{y}(1-\hat{y}) \cdot 1 = (\hat{y}-y)\cdot 1.$$