## pytorch fundamentals

typename = torch.tensor( , )

| type | Scaler | vector | matrix | tensor |
|---|---|---|---|---|
| ex) | 7 | [7,7] | [7,8],[9,10] | [[[1,2,3],[3,6,9],[2,4,5]]] |
| .ndim | 0 | 1 | 2 | 3 |
| .shape | [ ] | [2] | [2,2] | [1,3,3] |
| .item() | | | | |
| visualization | 7 | $\begin{bmatrix}7\\4\end{bmatrix}$ [7 4] | $\begin{bmatrix}7&10\\4&3\\5&1\end{bmatrix}$ | $\begin{bmatrix}[7\ 4]&[0\ 1]\\ [1\ 9]&[2\ 3]\\ [5\ 6]&[8\ 8]\end{bmatrix}$ |

torch.size( )

random_tensor = torch.rand(size=(r,c)) ... rand produces floats uniformly distributed across 0-1

.zeros

.ones

range_tensor = torch.arange(start, end, step)

tensor_like_w_zeros = torch.zeros_like(input = range_tensor) //creates tensor like range_tensor (same shape)

float32, float16

track gradients? yes or no?

tensor datatypes: t = torch.tensor( , dtype=None, device=None, requires_grad=False)

("cpu", "cuda" what device is tensor on?)

conversion between datatypes: float_16_tensor = float_32_tensor.type(torch.float16)

getting info from tensors: .dtype .shape .device

tensor manipulation: tensor +/-/*/ 10 //operates 10 on each index

torch. $\overset{mul}{\text{add}}$ (tensor, 10)

tensor · tensor //element wise multiplication

matrix multiplication torch. $\overset{.mm}{\text{matmul}}$ (tensor, tensor)

transpose tensor.T

tensor aggregation : .min() .max() .mean() .sum() //returns values

requires tensor of float 32

.argmin() .argmax() //returns index of values ex) tensor[0]

to fix dimension issues

will work on an element w/ xy elements

a = b.reshape(x,y)

a = b.view(x,y) //returns view of original tensor in different shape but shares same data as original tensor

torch.stack([inputs ], dim= ) //combines multiple tensors on top of each other or side by side

.squeeze() //removes all single dimensions from a target tensor

.unsqueeze(dim=x) //adds single dimension at dimension x

2nd dimension first, 0th dimension 2nd, 1st dimension 3rd

x.permute(new dimensions ex) (2,0,1)) //returns a view of original input with dimensions permuted (rearranged) indexing: ex) x: torch.arange[1,10].reshape(1,3,3) ... X[0][0] : tensor([1,2,3])

: will select all of a target dimension ... X[0][:][0] = tensor([1,4,7])

pytorch tensors ? numpy: torch.from_numpy(ndarray) // numpy data → pytorch tensor

torch.Tensor.numpy() // pytorch tensor → numpy

ex) import torch ... import numpy as np ... array = $\overbrace{\text{np.arange(1.0,8.0)}}^{\text{default is float64}}$ ... tensor = torch.from_numpy(array).$\overbrace{\text{type(torch.float32)}}^{\text{convert to tensor}}$

.print(tensor_A == tensorB)

reproducibility: ——— RANDOM_SEED=42 ... torch.manual_seed(RANDOM_SEED)

check for gpu access on pytorch: import torch ... torch.cuda.is_available()

setup device agnostic code: device = "cuda" if ↓ else "cpu"

count number of devices: torch.cuda.device_count()

putting tensors and models on gpus: tensor_on_gpu = tensor.to(device) — when instantiated, defaults to cpu

└ moving back to cpu: tensor_back_on_cpu = tensor_on_gpu().numpy() — numpy needs cpu