pytorch fundamentals

typename = torch.tensor( )

type	Scalar	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	[7 7]	[7 10; 9 3]	[[7 4] [0 1]; [1 9] [2 3]; [5 6] [8 8]]

torch.Size( )

random_tensor = torch.rand(size=(3,4)) ... rand produces floats uniformly distributed across 0-1

.zeros
.ones

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

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

tensor datatypes: t = torch.tensor(, dtype=None, device=None, requires_grad=False)
(float32, float16; "cpu", "cuda"; tracks gradients yes or no?)

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.add(tensor, 10)

tensor * tensor //element wise multiplication

matrix multiplication: torch.matmul(tensor, tensor) (.mm)

transpose: tensor.T

tensor aggregation: .min() .max() .mean() .sum() (requires tensor of float32) //returns values

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

fix dimension issues:

a = b.reshape(x,y) (will correct or error if x*y elements)

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

torch.stack([input,], 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

x.permute(new dimensions order)(2,0,1) //returns view of original input with dimensions permuted (reordered)

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 array → pytorch tensor

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

ex) import torch ... import numpy as np ... array = np.ones(1.0, 8.0) (default is float64) ... tensor = torch.from_numpy(array).type(torch.float32) (converts to tensor)

reproducibility: .min(tensorA == tensorB) ——— RANDOM_SEED = 42 ... torch.manual_seed(RANDOM_SEED)

check for gpu access with 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 gpu's: 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)