Machine Learning Preliminary

COMP4901Y

Binhang Yuan

Audit Policy

RELAXED SYSTEM LAB

- You are always welcome to come to my class or view the online resource;
- If you want an audit credit in your HKUST transcript:
  - Get **60% on the final exam**.
  - No mid-term or homework is required.

Linear Algebra

Some material in this section is modified from
https://courses.d2l.ai/berkeley-stat-157

Scalars

RELAXED
SYSTEM LAB

- Sample operations

$$c = a + b$$

$$c = a \cdot b$$

$$c = \sin a$$

- Length

$$|a| = \begin{cases} \text{a} & if \;\; a > 0 \\ -a & otherwise \end{cases}$$

$$|a + b| \leq |a| + |b|$$

$$|a \cdot b| = |a| \cdot |b|$$

Vector

RELAXED SYSTEM LAB

- Vector in n-dimensions $\boldsymbol{a} \in \mathbb{R}^n$, $\mathbf{a}= \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix}$
- Sample operations:

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} \ \ where\ c_i = a_i + b_i$$

$$\boldsymbol{c} = \alpha \cdot \boldsymbol{b} \ \ where\ c_i = \alpha b_i$$

$$\boldsymbol{c} = \sin \boldsymbol{a} \ \ where\ c_i = \sin a_i$$

Vector

RELAXED
SYSTEM LAB

- Some properties of vector addition:
  - Commutative:

$$\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{b} + \boldsymbol{a}, \qquad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$$

  - Associative:

$$(\boldsymbol{a} + \boldsymbol{b}) + \boldsymbol{c} = \boldsymbol{a} + (\boldsymbol{b} + \boldsymbol{c}), \qquad \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^n$$

  - Distibutive:

$$\alpha(\boldsymbol{a} + \boldsymbol{b}) = \alpha\boldsymbol{a} + \alpha\boldsymbol{b}, \qquad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$$

Vector

RELAXED SYSTEM LAB

- Euclidean Length

$$\|\boldsymbol{a}\|_2 = \left[\sum_{i=1}^{m} a_i^2\right]^{\frac{1}{2}}$$

$$\|\boldsymbol{a}\| \geq 0 \; \forall \; \boldsymbol{a}$$

$$\|\boldsymbol{a} + \boldsymbol{b}\| \leq \|\boldsymbol{a}\| + \|\boldsymbol{b}\|$$

$$\|\alpha \cdot \boldsymbol{b}\| \leq |\alpha| \|\boldsymbol{b}\|$$

- p-norm

$$\|\boldsymbol{a}\|_p = \left[\sum_{i=1}^{m} a_i^p\right]^{\frac{1}{p}}$$

Vector

RELAXED SYSTEM LAB

- The span of a set of vectors:

$$\text{span}\{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_k}\} = \{\alpha_1 \boldsymbol{a_1} + \alpha_2 \boldsymbol{a_2} + \cdots + \alpha_k \boldsymbol{a_k} \,|\, \alpha_k \in \mathbb{R},\ \boldsymbol{a_k} \in \mathbb{R}^n\}$$

- Linear independence:

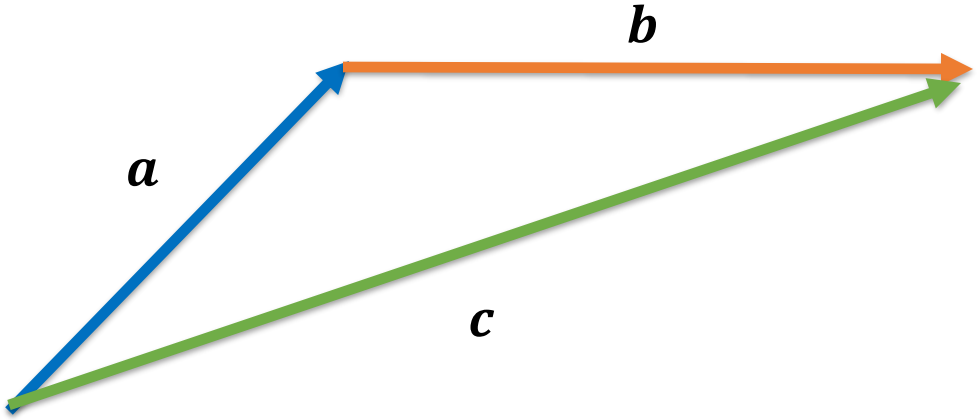
$$\alpha_1 \boldsymbol{a_1} + \alpha_2 \boldsymbol{a_2} + \cdots + \alpha_k \boldsymbol{a_k} = \boldsymbol{0} \Rightarrow\ \alpha_i = 0, \forall i$$

- How does k compare to n, the vector dimension?

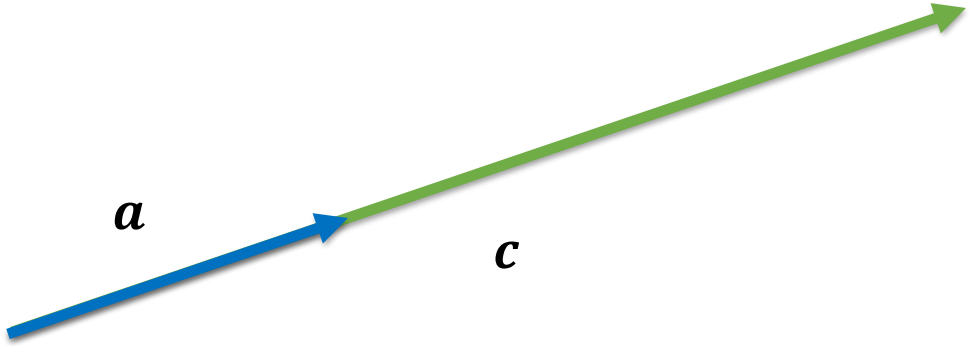
$$k \leq n$$

Vector

RELAXED SYSTEM LAB



$$c = a + b$$



$$c = \alpha \cdot a$$

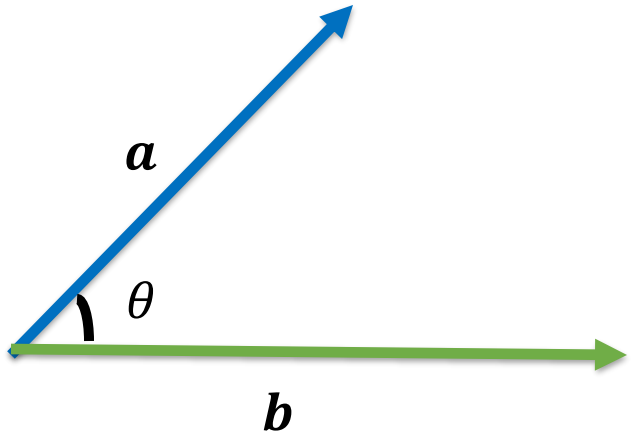
Mathematician's **'parallel for all do'**

Vectors

RELAXED SYSTEM LAB

- Inner product

$$\boldsymbol{a}^T\boldsymbol{b} = \sum_i a_i b_i = \|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\| \cdot \cos\theta$$



- Orthogonality

$$\boldsymbol{a}^T\boldsymbol{b} = \sum_i a_i b_i = 0$$

  - If we have two vectors that are orthogonal with a third, their linear combination is, too.

Matrices

RELAXED SYSTEM LAB

- Matrix in m, n-dimensions: $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

$$\mathbf{A} = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix}$$

- Transpose of matrix:

$$\mathbf{A}^{\mathrm{T}} = \begin{bmatrix} A_{11} & \cdots & A_{m1} \\ \vdots & \ddots & \vdots \\ A_{1n} & \cdots & A_{mn} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

$$(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A}^T + \boldsymbol{B}^T, \forall \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m \times n}$$

Matrices

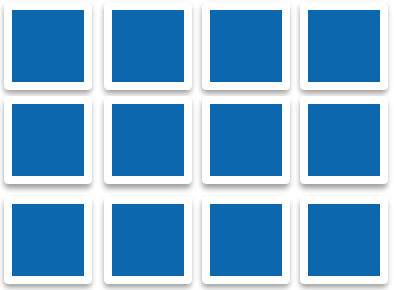
RELAXED
SYSTEM LAB

- Simple operations

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} \;\; where \; C_{ij} = A_{ij} + B_{ij}$$

$$\boldsymbol{C} = \alpha \cdot \boldsymbol{B} \;\; where \; C_{ij} = \alpha B_{ij}$$

$$\boldsymbol{C} = \sin \boldsymbol{A} \;\; where \; C_{ij} = \sin A_{ij}$$



Matrices

RELAXED SYSTEM LAB

- Some properties of matrix addition:
  - Commutative:

$$\boldsymbol{A} + \boldsymbol{B} = \boldsymbol{B} + \boldsymbol{A}, \qquad \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m \times n}$$

  - Associative:

$$(\boldsymbol{A} + \boldsymbol{B}) + \boldsymbol{C} = \boldsymbol{A} + (\boldsymbol{B} + \boldsymbol{C}), \qquad \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{m \times n}$$

  - Distibutive:

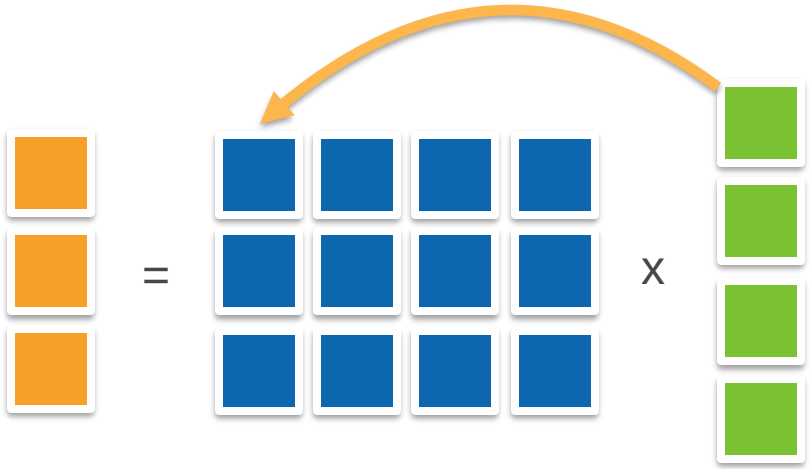
$$\alpha(\boldsymbol{A} + \boldsymbol{B}) = \alpha\boldsymbol{A} + \alpha\boldsymbol{B}, \qquad \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m \times n}$$

Matrices

RELAXED SYSTEM LAB

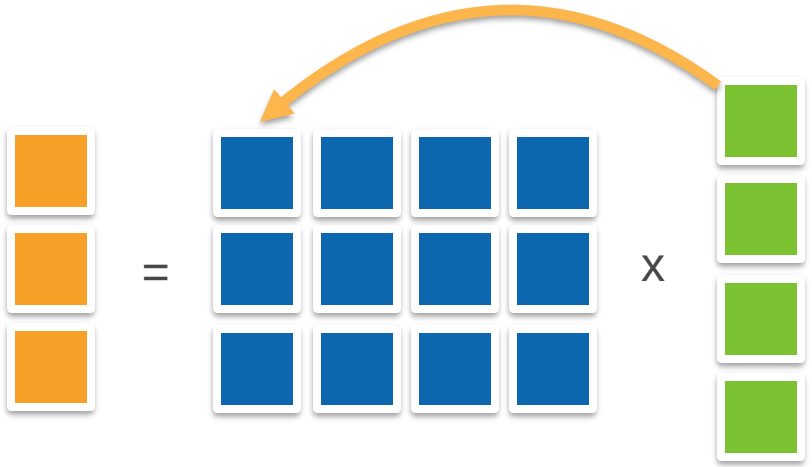
- Multiplications (matrix-vector), $\boldsymbol{c} = \boldsymbol{Ab}, \boldsymbol{c} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{B} \in \mathbb{R}^n$

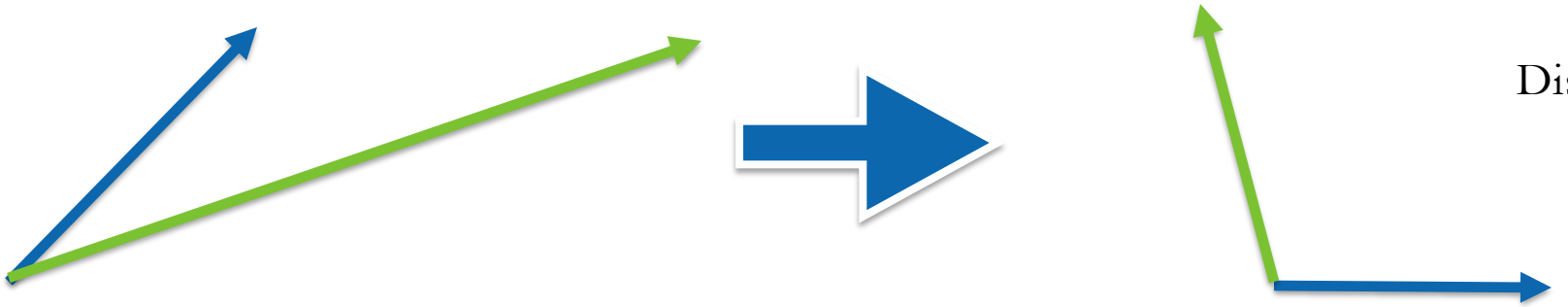
$$\begin{bmatrix} c_1 \\ \cdots \\ c_m \end{bmatrix} = \boldsymbol{c} = \boldsymbol{Ab} = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix} \quad where\ c_i = \sum_{j=1}^{n} A_{ij} b_j$$



Matrices

RELAXED
SYSTEM LAB





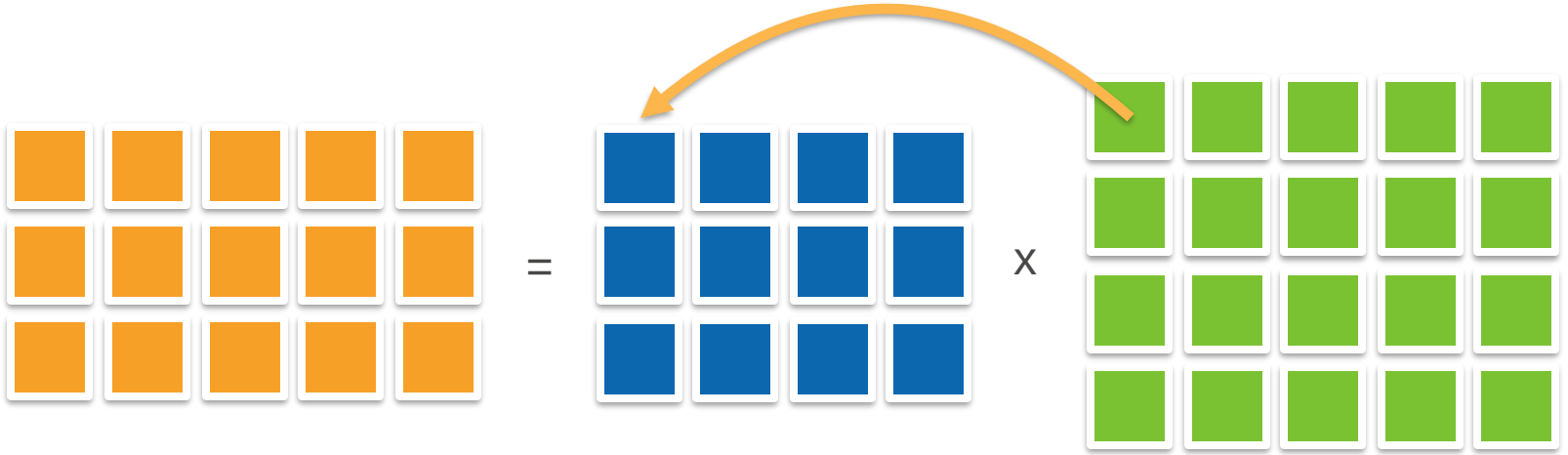
Distort the space.

Matrices

RELAXED SYSTEM LAB

- Multiplications (matrix-matrix) $\boldsymbol{C} = \boldsymbol{AB}, \boldsymbol{C} \in \mathbb{R}^{m \times p}, \boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{B} \in \mathbb{R}^{n \times p}$

$$\begin{bmatrix} C_{11} & \cdots & C_{1p} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mp} \end{bmatrix} = \boldsymbol{C} = \boldsymbol{AB} = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} \begin{bmatrix} B_{11} & \cdots & A_{1p} \\ \vdots & \ddots & \vdots \\ B_{n1} & \cdots & A_{np} \end{bmatrix} \text{where } C_{ik} = \sum_{j=1}^{n} A_{ij} B_{jk}$$



Matrices

RELAXED SYSTEM LAB

- Some properties of matrix multiplication:
  - Non-commutative!

$$\boldsymbol{AB} \neq \boldsymbol{BA}$$

  - Associative:

$$(\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{A}(\boldsymbol{BC}), \qquad \forall \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$$

$$\alpha(\boldsymbol{AB}) = (\alpha\boldsymbol{A})\boldsymbol{B}, \qquad \forall \boldsymbol{A}, \boldsymbol{B}$$

  - Distibutive:

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AB}, \qquad \forall \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$$

  - Transpose:

$$(\boldsymbol{AB})^T = \boldsymbol{B}^T \boldsymbol{A}^T$$

Matrix

RELAXED SYSTEM LAB

- Norm

$$\|\boldsymbol{A}\| = \sup\left\{\frac{\|\boldsymbol{Ax}\|_p}{\|\boldsymbol{x}\|_p}, \forall\, \boldsymbol{x} \neq 0\right\}$$

  - Interpretation: how much can the mapping induced by $\boldsymbol{A}$ can stretch vectors?
  - Choices depend on how to measure the length of vectors:
- Popular norms
  - Frobenius norm (p=2):

$$\|\boldsymbol{A}\|_{Frob} = \left[\sum_{ij} A_{ij}^2\right]^{\frac{1}{2}}$$

  - Max norm (p=∞):

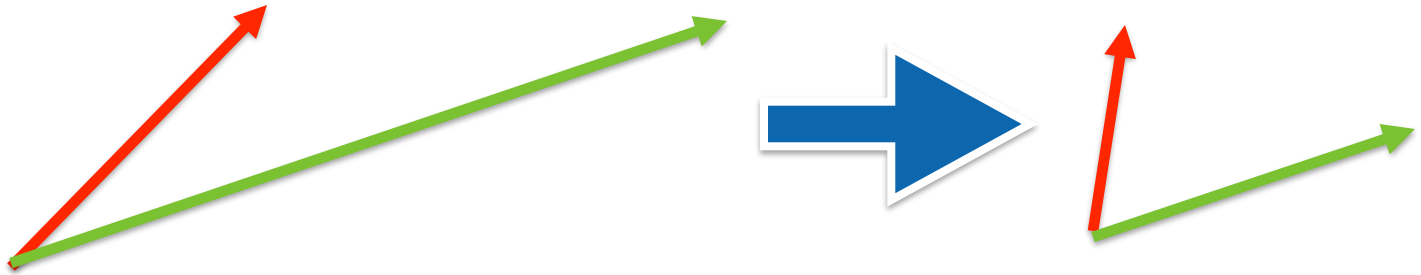
$$\|\boldsymbol{A}\|_{Max} = \max|a_{ij}|$$

Matrix

RELAXED SYSTEM LAB

- Eigenvectors and egienvalue
  - Vectors that are not changed by the matrix ($\boldsymbol{x}$ is the vector, λ is the eginvalue):

$$\boldsymbol{Ax} = \lambda \boldsymbol{x}$$



  - For symmetric matrices, we can always find the eigenvalue and eigenvector.

Special Matrices $\boldsymbol{A} \in \mathbb{R}^{n \times n}$

RELAXED SYSTEM LAB

- Symmetric matrix: $\boldsymbol{A}^T = \boldsymbol{A}$

$$A_{ij} = A_{ji}$$

- Antisymmetric matrix: $\boldsymbol{A}^T = -\boldsymbol{A}$

$$A_{ij} = -A_{ji}$$

- Positive definite:

$$\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geq 0, \qquad \forall \boldsymbol{x}$$

Special Matrices

RELAXED SYSTEM LAB

- Orthogonal Matrices:
  - All rows of the matrix are orthogonal to each other;
  - All rows of the matrix have unit length.
  - Rewrite in matrix form:

$$\sum_j U_{ij}U_{kj} = \delta_{ik}$$

$$\boldsymbol{UU^T = I}$$

- Permutation Matrices:
  - There is only one 1 in each row or column:

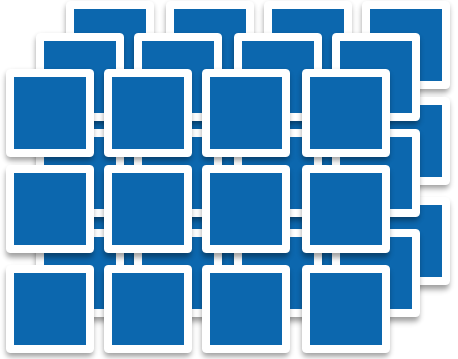
$$P_{ij} = \begin{cases} 1 \; if \; and \; only \; if \; j = \pi(i) \\ 0. \qquad\qquad\qquad otherwise \end{cases}$$

Tensor

RELAXED
SYSTEM LAB

- A tensor is a collection of numbers labelled by indices.

- The rank of a tensor is the number of indices required to specify an entry in the tensor:
  - A vector is a rank–1 tensor;
  - A matrix is a rank–2 tensor.



Tensor

RELAXED SYSTEM LAB

- Einstein summation convention:
  - Each index can appear at most twice in any term.
  - Repeated indices are implicitly summed over.
  - Each term must contain identical non-repeated indices.

$$M_{ij}v_j \equiv \sum_i M_{ij}v_j$$

Matrix multiplication between matrix M and vector v

$$M_{ij}u_jv_j$$

The index j appears three times in the first term.

$$T_{ijk}u_k + M_{ip}$$

The first term contains the non-repeated index j whereas the second term contains p.

Tensors in PyTorch

Creating Tensors

RELAXED SYSTEM LAB

- Tensors are the central data abstraction in PyTorch.

Code	Output

`import torch`
`import math`
 	
`x = torch.empty(3, 4)`
`print(type(x))`
`print(x)`
	`<class 'torch.Tensor'>`
	`tensor([[-1.9609e-04,  4.5654e-41,  4.4115e-04,  0.0000e+00],`
	`[ 8.0387e+26,  4.5654e-41,  8.9824e-06,  0.0000e+00],`
	`[ 1.3431e-14,  0.0000e+00,  0.0000e+00,  0.0000e+00]])`

Creating Tensors

RELAXED SYSTEM LAB

Code	Output

`import torch`
`import math`
 	
`x = torch.empty(3, 4)`
`print(type(x))`
`print(x)`
	`<class 'torch.Tensor'>`
	`tensor([[-1.9609e-04,  4.5654e-41,  4.4115e-04,  0.0000e+00],`
	`[ 8.0387e+26,  4.5654e-41,  8.9824e-06,  0.0000e+00],`
	`[ 1.3431e-14,  0.0000e+00,  0.0000e+00,  0.0000e+00]])`

- Create a tensor using one of the numerous factory methods attached to the torch module.
- The tensor itself is 2-dimensional, having 3 rows and 4 columns.
- The type of the object returned is torch.Tensor, which is an alias for torch.FloatTensor; by default, PyTorch tensors are populated with 32-bit floating point numbers.
- You will probably see some random-looking values when printing your tensor.
- The torch.empty() call allocates memory for the tensor, but does not initialize it with any values.
- What you're seeing is whatever was in memory at the time of allocation.

Creating Tensors

RELAXED SYSTEM LAB

- Initialize your tensor with some value.

Code	Output

`zeros = torch.zeros(2, 3)`
`print(zeros)`
 	
`ones = torch.ones(2, 3)`
`print(ones)`
 	
`torch.manual_seed(1729)`
`random = torch.rand(2, 3)`
`print(random)`
	`tensor([[0., 0., 0.],`
	`[0., 0., 0.]])`
	`tensor([[1., 1., 1.],`
	`[1., 1., 1.]])`
	`tensor([[0.3126, 0.3791, 0.3087],`
	`[0.0736, 0.4216, 0.0691]])`

Creating Tensors

RELAXED SYSTEM LAB

- Manually setting your random number generator's seed assures reproducibility.

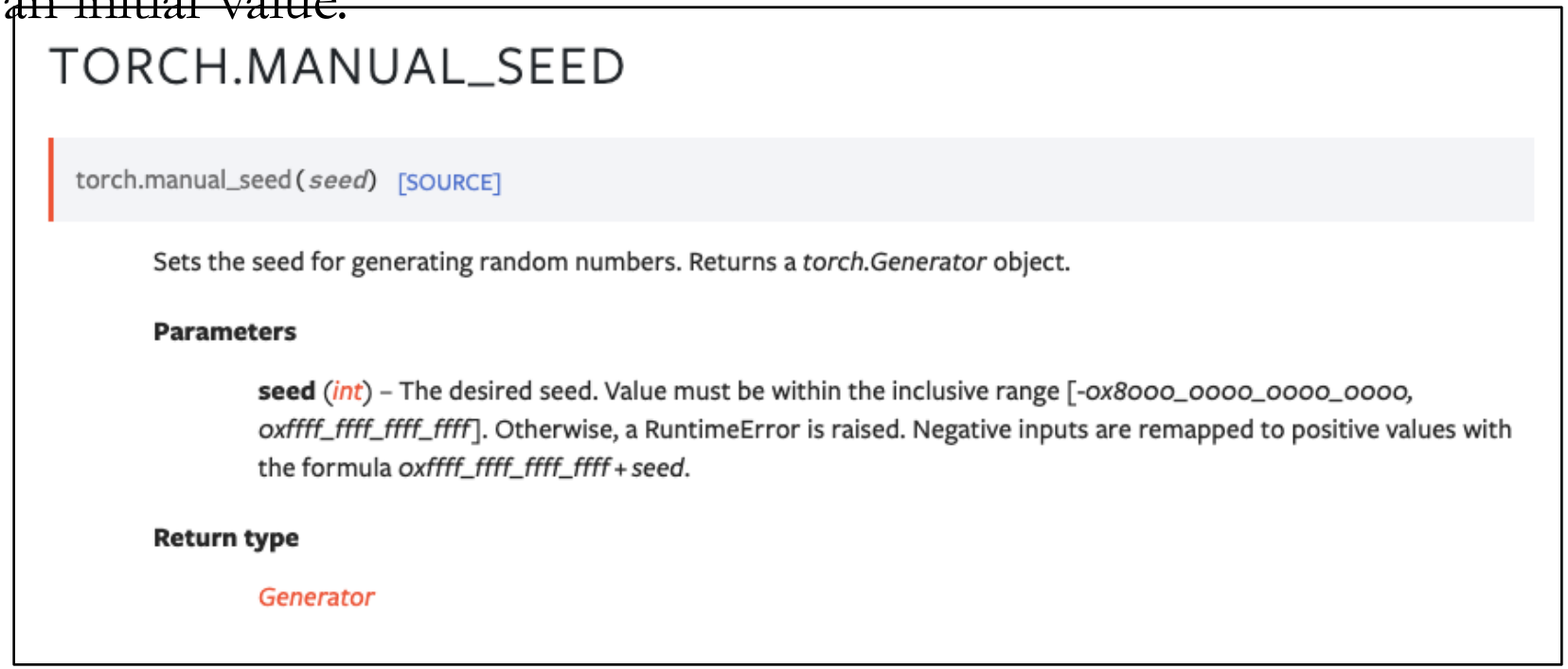
Code	Output

`torch.manual_seed(1729)`
`random1 = torch.rand(2, 3)`
`print(random1)`
 	
`random2 = torch.rand(2, 3)`
`print(random2)`
 	
`torch.manual_seed(1729)`
`random3 = torch.rand(2, 3)`
`print(random3)`
 	
`random4 = torch.rand(2, 3)`
`print(random4)`
	`tensor([[0.3126, 0.3791, 0.3087],`
	`[0.0736, 0.4216, 0.0691]])`
	`tensor([[0.2332, 0.4047, 0.2162],`
	`[0.9927, 0.4128, 0.5938]])`
	`tensor([[0.3126, 0.3791, 0.3087],`
	`[0.0736, 0.4216, 0.0691]])`
	`tensor([[0.2332, 0.4047, 0.2162],`
	`[0.9927, 0.4128, 0.5938]])`

Creating Tensors

RELAXED SYSTEM LAB

- A pseudorandom number generator is a ***deterministic*** random bit generator:
  - An algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.
  - The generated sequence is not truly random, it is completely determined by ~~an initial value.~~



TORCH.MANUAL_SEED

torch.manual_seed(*seed*) [SOURCE]

Sets the seed for generating random numbers. Returns a *torch.Generator* object.

Parameters

seed (*int*) – The desired seed. Value must be within the inclusive range [*-0x8000_0000_0000_0000*, *0xffff_ffff_ffff_ffff*]. Otherwise, a RuntimeError is raised. Negative inputs are remapped to positive values with the formula *0xffff_ffff_ffff_ffff + seed*.

Return type

Generator

Creating Tensors

RELAXED SYSTEM LAB

- torch.shape contains a list of the extent of each dimension of a tensor.

Code	Output

`x = torch.empty(2, 2, 3)`
`print(x.shape)`
 	
`empty_like_x =`
`torch.empty_like(x)`
`print(empty_like_x.shape)`
 	
`zeros_like_x =`
`torch.zeros_like(x)`
`print(zeros_like_x.shape)`
 	
`ones_like_x =`
`torch.ones_like(x)`
`print(ones_like_x.shape)`
 	
`rand_like_x =`
`torch.rand_like(x)`
`print(rand_like_x.shape)`
	`torch.Size([2, 2, 3])`
	
	`torch.Size([2, 2, 3])`
	
	`torch.Size([2, 2, 3])`
	
	`torch.Size([2, 2, 3])`
	
	`torch.Size([2, 2, 3])`

Creating Tensors

RELAXED SYSTEM LAB

- Create a tensor by specifying its data directly from a Python collection.

Code	Output

`some_constants = torch.tensor([[3.1415926, 2.71828],`
`[1.61803, 0.0072897]])`
`print(some_constants)`
 	
`some_integers = torch.tensor((2, 3, 5, 7, 11, 13, 17, 19))`
`print(some_integers)`
 	
`more_integers = torch.tensor(((2, 4, 6), [3, 6, 9]))`
`print(more_integers)`
	`tensor([[3.1416, 2.7183],`
	`[1.6180, 0.0073]])`
	`tensor([ 2,  3,  5,  7, 11, 13, 17, 19])`
	`tensor([[2, 4, 6],`
	`[3, 6, 9]])`

Creating Tensors

RELAXED SYSTEM LAB

- Setting the datatype of a tensor.

Code	Output

`a = torch.ones((2, 3), dtype=torch.int16)`
`print(a)`
 	
`b = torch.rand((2, 3), dtype=torch.float64) * 20.`
`print(b)`
 	
`c = b.to(torch.int32)`
`print(c)`
	`tensor([[1, 1, 1],`
	`[1, 1, 1]], dtype=torch.int16)`
	`tensor([[ 0.9956, 1.4148, 5.8364],`
	`[11.2406, 11.2083, 11.6692]], dtype=torch.float64)`
	`tensor([[ 0, 1, 5],`
	`[11, 11, 11]], dtype=torch.int32)`

Creating Tensors

- Available data types include:
  - torch.bool
  - torch.int8
  - torch.uint8
  - torch.int16
  - torch.int32
  - torch.int64
  - torch.half
  - torch.float
  - torch.double
  - torch.bfloat

32-bit floating point	`torch.float32` or `torch.float`	`torch.FloatTensor`	`torch.cuda.FloatTensor`
64-bit floating point	`torch.float64` or `torch.double`	`torch.DoubleTensor`	`torch.cuda.DoubleTensor`
16-bit floating point 1	`torch.float16` or `torch.half`	`torch.HalfTensor`	`torch.cuda.HalfTensor`
16-bit floating point 2	`torch.bfloat16`	`torch.BFloat16Tensor`	`torch.cuda.BFloat16Tensor`

8-bit integer (unsigned)	`torch.uint8`	`torch.ByteTensor`	`torch.cuda.ByteTensor`
8-bit integer (signed)	`torch.int8`	`torch.CharTensor`	`torch.cuda.CharTensor`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.ShortTensor`	`torch.cuda.ShortTensor`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.IntTensor`	`torch.cuda.IntTensor`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.LongTensor`	`torch.cuda.LongTensor`

Math & Logic with PyTorch Tensors

RELAXED SYSTEM LAB

- Basic arithmetic: how tensors interact with simple scalars.

Code	Output

`ones = torch.zeros(2, 2) + 1`
`twos = torch.ones(2, 2) * 2`
`threes = (torch.ones(2, 2) * 7`
`- 1) / 2`
`fours = twos ** 2`
`sqrt2s = twos ** 0.5`
 	
`print(ones)`
`print(twos)`
`print(threes)`
`print(fours)`
`print(sqrt2s)`
	`tensor([[1., 1.],`
	`[1., 1.]])`
	`tensor([[2., 2.],`
	`[2., 2.]])`
	`tensor([[3., 3.],`
	`[3., 3.]])`
	`tensor([[4., 4.],`
	`[4., 4.]])`
	`tensor([[1.4142, 1.4142],`
	`[1.4142, 1.4142]])`

Math & Logic with PyTorch Tensors

RELAXED SYSTEM LAB

- Basic arithmetic operations between two tensors.

Code	Output

`powers2 = twos ** torch.tensor([[1, 2], [3, 4]])`
`print(powers2)`
 	
`fives = ones + fours`
`print(fives)`
 	
`dozens = threes * fours`
`print(dozens)`
	`tensor([[ 2.,  4.],`
	`[ 8., 16.]])`
	`tensor([[5., 5.],`
	`[5., 5.]])`
	`tensor([[12., 12.],`
	`[12., 12.]])`

Math & Logic with PyTorch Tensors

RELAXED SYSTEM LAB

- Tensor broadcasting:
  - Perform an operation between tensors that have similarities in their shapes;
  - E.g., a common example:
    - Multiply a tensor of learning weights by a batch of input tensors;
    - Apply the operation to each instance in the batch separately;
    - Return a tensor of identical shape.
- Four rule for broadcasting:
  - Each tensor must have at least one dimension (no empty tensors).
  - Comparing the dimension sizes of the two tensors, going ***from last to first***:
    - Each dimension must be equal, or
    - One of the dimensions must be of size 1, or
    - The dimension does not exist in one of the tensors.

Math & Logic with PyTorch Tensors

RELAXED SYSTEM LAB

- Examples allow broadcasting.

Code	Output		
a = torch.ones(4, 3, 2) # 3rd & 2nd dims identical to a, dim 1 absent b = a * torch.rand(3, 2) print(b) # 3rd dim = 1, 2nd dim identical to a c = a * torch.rand(3, 1) print(c) # 3rd dim identical to a, 2nd dim = 1 d = a * torch.rand(1, 2) print(d)	tensor([[[0.6493, 0.2633], [0.4762, 0.0548], [0.2024, 0.5731]], [[0.6493, 0.2633], [0.4762, 0.0548], [0.2024, 0.5731]], [[0.6493, 0.2633], [0.4762, 0.0548], [0.2024, 0.5731]], [[0.6493, 0.2633], [0.4762, 0.0548], [0.2024, 0.5731]]])	tensor([[[0.7191, 0.7191], [0.4067, 0.4067], [0.7301, 0.7301]], [[0.7191, 0.7191], [0.4067, 0.4067], [0.7301, 0.7301]], [[0.7191, 0.7191], [0.4067, 0.4067], [0.7301, 0.7301]], [[0.7191, 0.7191], [0.4067, 0.4067], [0.7301, 0.7301]]])	tensor([[[0.6276, 0.7357], [0.6276, 0.7357], [0.6276, 0.7357]], [[0.6276, 0.7357], [0.6276, 0.7357], [0.6276, 0.7357]], [[0.6276, 0.7357], [0.6276, 0.7357], [0.6276, 0.7357]], [[0.6276, 0.7357], [0.6276, 0.7357], [0.6276, 0.7357]]])

Math & Logic with PyTorch Tensors

RELAXED SYSTEM LAB

- Examples attempt at broadcasting that will fail.

Code

```
a = torch.ones(4, 3, 2)

b = a * torch.rand(4, 3)    # dimensions must match last-to-first

c = a * torch.rand(   2, 3) # both 3rd & 2nd dims different

d = a * torch.rand((0, ))   # can't broadcast with an empty tensor
```

Altering Tensors in Place

RELAXED SYSTEM LAB

- Most binary operations on tensors will return a third, new tensor that takes a region of memory distinct from the other tensors.
- Most of the math functions have a version with an appended underscore (_) that will alter a tensor in place.

Code

```
a = torch.tensor([0, math.pi / 4, math.pi / 2, 3 * math.pi / 4])
print('a:')
print(a)
print(torch.sin(a))   # this operation creates a new tensor in memory
print(a)              # a has not changed

b = torch.tensor([0, math.pi / 4, math.pi / 2, 3 * math.pi / 4])
print('\nb:')
print(b)
print(torch.sin_(b))  # note the underscore
print(b)              # b has changed
```

Altering Tensors in Place

RELAXED SYSTEM LAB

- Set up an out argument to specify a tensor to receive the output. If the out tensor is the correct shape and dtype, this can happen without a new memory allocation.

Code	Output

`a = torch.rand(2, 2)`
`b = torch.rand(2, 2)`
`c = torch.zeros(2, 2)`
`old_id = id(c)`
 	
`print(c)`
`d = torch.matmul(a, b, out=c)`
`print(c)` `# contents of c have changed`
 	
`assert c is d` `# test c & d are same object, not just containing equal values`
`assert id(c) == old_id` `# make sure that our new c is the same object as the old one`
 	
`torch.rand(2, 2, out=c)` `# works for creation too!`
`print(c)` `# c has changed again`
`assert id(c) == old_id` `# still the same object!`
	`tensor([[0., 0.],`
	`[0., 0.]])`
	`tensor([[0.3653, 0.8699],`
	`[0.2364, 0.3604]])`
	`tensor([[0.0776, 0.4004],`
	`[0.9877, 0.0352]])`

Copying Tensors

RELAXED SYSTEM LAB

- As with any object in Python, assigning a tensor to a variable makes the variable a label of the tensor, and does not copy it.
- If you want a separate copy of the data to work on? The clone() method is there for you.

Code	Output

`a = torch.ones(2, 2)`
`b = a.clone()`
 	
`assert b is not a  # different objects in memory...`
`print(torch.eq(a, b))  # ...but still with the same contents!`
 	
`a[0][1] = 561  # a changes...`
`print(b)  # ...but b is still all ones`
	`tensor([[True, True],`
	`[True, True]])`
	`tensor([[1., 1.],`
	`[1., 1.]])`

Moving to GPU

RELAXED SYSTEM LAB

- Define string or torch device handle to move the tensor to GPU.
- To do computation involving two or more tensors, all of the tensors must be on the ***same*** device.

Code	Output

`if torch.cuda.is_available():`
`  my_device = torch.device('cuda')`
`  x = torch.rand(2, 2,  device='cuda')`
`  print(gpu_rand)`
`else:`
`  my_device = torch.device('cpu')`
`print('Device: {}'.format(my_device))`
 	
`y = torch.rand(2, 2, device=my_device)`
`print(y)`
	`tensor([[0.3344, 0.2640],`
	`  [0.2119, 0.0582]], device='cuda:0')`
	
	`Device: cuda`
	`tensor([[0.0024, 0.6778],`
	`  [0.2441, 0.6812]], device='cuda:0')`

Manipulating Tensor Shapes - Squeeze

RELAXED SYSTEM LAB

torch.squeeze(*input*, *dim=None*) → Tensor

Returns a tensor with all specified dimensions of `input` of size *1* removed.

For example, if *input* is of shape: $(A \times 1 \times B \times C \times 1 \times D)$ then the *input.squeeze()* will be of shape: $(A \times B \times C \times D)$.

When `dim` is given, a squeeze operation is done only in the given dimension(s). If *input* is of shape: $(A \times 1 \times B)$, `squeeze(input, 0)` leaves the tensor unchanged, but `squeeze(input, 1)` will squeeze the tensor to the shape $(A \times B)$.

• NOTE

The returned tensor shares the storage with the input tensor, so changing the contents of one will change the contents of the other.

• WARNING

If the tensor has a batch dimension of size 1, then *squeeze(input)* will also remove the batch dimension, which can lead to unexpected errors. Consider specifying only the dims you wish to be squeezed.

The squeeze() method has the in-place versions squeeze_().

Manipulating Tensor Shapes - Unsqueeze

RELAXED SYSTEM LAB

torch.unsqueeze(*input*, *dim*) → Tensor

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

The returned tensor shares the same underlying data with this tensor.

A `dim` value within the range `[-input.dim() - 1, input.dim() + 1)` can be used. Negative `dim` will correspond to `unsqueeze()` applied at `dim = dim + input.dim() + 1`.

Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the index at which to insert the singleton dimension

The unsqueeze() method has the in-place versions unsqueeze_().

Manipulating Tensor Shapes - Reshape

RELAXED SYSTEM LAB

torch.reshape(*input*, *shape*) → Tensor

Returns a tensor with the same data and number of elements as `input`, but with the specified shape. When possible, the returned tensor will be a view of `input`. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

See `torch.Tensor.view()` on when it is possible to return a view.

A single dimension may be -1, in which case it's inferred from the remaining dimensions and the number of elements in `input`.

Parameters

- **input** (*Tensor*) – the tensor to be reshaped
- **shape** (*tuple of int*) – the new shape

reshape() will return a *view* on the tensor to be changed: a separate tensor object looking at the same underlying region of memory. That means any change made to the source tensor will be reflected in the view on that tensor, unless you clone() it.

References

RELAXED SYSTEM LAB

- https://d2l.ai/
- https://www.dr-qubit.org/teaching/summation_delta.pdf
- https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html