Background

Background

- Linear algebra
- Probabilities
- Tensors
- Introduction to pytorch

Linear algebra and gradients

Overview

- Notation: vector, matrix
- Definition: vector and matrix operations
- Gradients and chain rule

What is a vector?

- An array of numbers
- Notation: $\mathbf{v}$
  - bold lower case
- Size: $\text{size}(\mathbf{v}) = n$
- Order: $\dim(\mathbf{v}) = 1$
- Indexing: $\mathbf{v}_i$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 4.3 \\ 1.2 \\ 9.9 \\ 2.3 \end{bmatrix}$$

$$\begin{bmatrix} 1.2 \\ 4.4 \end{bmatrix}$$

What is a matrix?

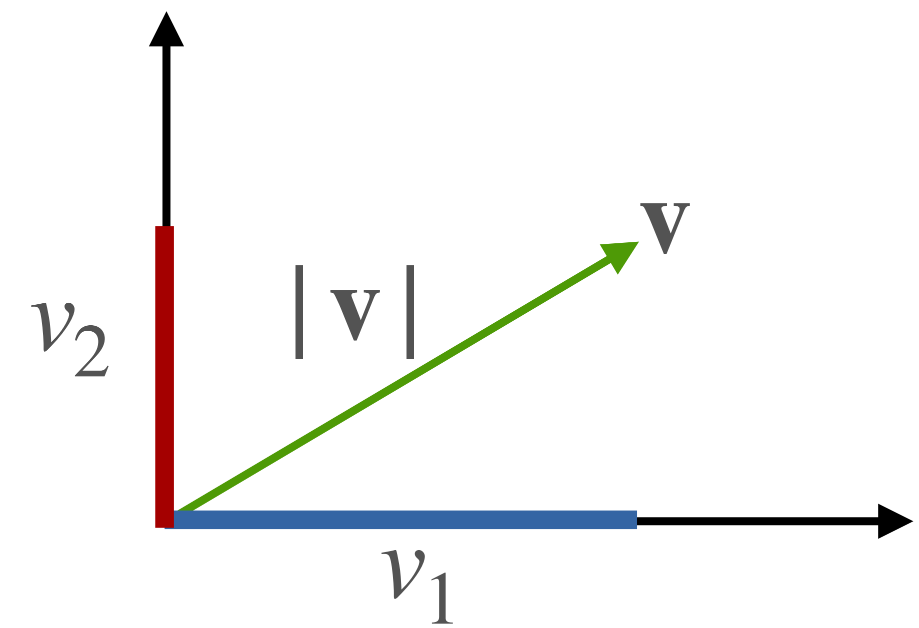
- An 2D array of numbers
- Notation: $\mathbf{M}$
  - bold upper case
- Size: $\text{size}(\mathbf{M}) = n \times m$
- Order: $\dim(\mathbf{M}) = 2$
- Indexing: $\mathbf{M}_{ij}$

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1.2 & 1.1 \\ 4.4 & 3.2 \end{bmatrix}$$

Vector norm

$$|\mathbf{v}| = \sqrt{\sum_{i}^{n} v_i^2}$$



Element wise operations

$$\mathbf{v} + \mathbf{w} = \begin{bmatrix} v_0 + w_0 \\ v_1 + w_1 \\ \cdots \\ v_n + w_n \end{bmatrix} \quad \mathbf{v} * \mathbf{w} = \begin{bmatrix} v_0 * w_0 \\ v_1 * w_1 \\ \cdots \\ v_n * w_n \end{bmatrix} \quad \cdots$$

$$\mathbf{v} - \mathbf{w} = \begin{bmatrix} v_0 - w_0 \\ v_1 - w_1 \\ \cdots \\ v_n - w_n \end{bmatrix} \quad \mathbf{v}/\mathbf{w} = \begin{bmatrix} v_0/w_0 \\ v_1/w_1 \\ \cdots \\ v_n/w_n \end{bmatrix} \quad \cdots$$

Inner product

$$\mathbf{v} \cdot \mathbf{w} = v_0 \cdot w_0 + v_1 \cdot w_1 + \ldots + v_n \cdot w_n$$

Outer product

$$\mathbf{v} \otimes \mathbf{w} = \begin{bmatrix} v_0 w_0 & v_0 w_1 & \cdots & v_0 w_n \\ v_1 w_0 & v_1 w_1 & \cdots & v_1 w_n \\ & & \cdots & \\ v_n w_0 & v_n w_1 & \cdots & v_n w_n \end{bmatrix}$$

Frobenius norm

$$|\mathbf{M}| = \sqrt{\sum_{i}^{n} \sum_{j}^{m} M_{ij}^2}$$

Transpose

$$\mathbf{M} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \Rightarrow \mathbf{M}^{\top} = \begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \\ a_{02} & a_{12} \end{bmatrix}$$

Matrix multiplication

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \\ b_{30} & b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{00} & \boxed{c_{01}} & c_{02} \\ c_{10} & c_{11} & c_{12} \end{bmatrix}$$

2x4 4x3 2x3

Matrix-vector multiplication

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \end{bmatrix} \begin{bmatrix} b_{00} \\ b_{10} \\ b_{20} \\ b_{30} \end{bmatrix} = \begin{bmatrix} c_{00} \\ c_{10} \end{bmatrix}$$

2x4 4 2

Vector operations

- Length / norm: $|\mathbf{v}|$
- Element-wise operations: $\mathbf{v} + \mathbf{w}, \mathbf{v} - \mathbf{w}, \ldots$
- Inner (dot) product: $\mathbf{v}^\top \mathbf{w} = \mathbf{v} \cdot \mathbf{w}$
- Outer product: $\mathbf{v}\mathbf{w}^\top = \mathbf{v} \otimes \mathbf{w}$

Matrix Operations

- Frobenius norm $|\mathbf{M}|$
- Transpose $\mathbf{M}^\top$
- Matrix multiplication $\mathbf{AB}$ $\mathbf{Av}$

Functions of vectors

- Definition: $f : \mathbf{x} \rightarrow \mathbf{y}$

Derivatives of vector valued functions

- Gradient of scaler function

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_0} & \frac{\partial f}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f}{\partial \mathbf{x}_n} \end{bmatrix}$$

scaler

- **Jacobian**: Derivative of vector $\mathbf{g}(\mathbf{x})$ by vector $\mathbf{x}$:

$$\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{g}}{\partial \mathbf{x}_0} & \frac{\partial \mathbf{g}}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{g}}{\partial \mathbf{x}_n} \end{bmatrix}$$

vector

Chain rule for scalar functions

- Nested functions: $f(g(x))$
    - where $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$
    - and $y = g(x)$
- Derivative: $$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(y)}{\partial y} \frac{\partial g(x)}{\partial x}$$

Chain rule for vector-valued functions

- Nested functions: $f(g(\mathbf{x}))$
  - where $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^p \rightarrow \mathbb{R}^m$
  - and $\mathbf{y} = g(\mathbf{x})$
- Derivative: $$\underbrace{\frac{\partial f(g(\mathbf{x}))}{\partial \mathbf{x}}}_{\text{n x p}} = \underbrace{\frac{\partial f(\mathbf{y})}{\partial \mathbf{y}}}_{\text{n x m}} \underbrace{\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}}_{\text{m x p}}$$

Summary

- Vector **v**
- Matrix **M**
- Gradients $\frac{\partial f(x)}{\partial x}$
- Chain rule $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(y)}{\partial y} \frac{\partial g(x)}{\partial x}$
- More reading
  [The Matrix Cookbook, Petersen and Pedersen 2012]

Probability, likelihood, sampling and expectation

Overview

- Probabilities and likelihood
- Sampling
- Linearity of expectation

What is a (discrete) distribution?

- Informal: bunch of positive numbers that some to one
- Distribution: $P : a \rightarrow [0,1]$
- Event: $a \in [0,\ldots,n-1]$
- Probability: $P(a)$
    - Chance of a occurring

What is a conditional probability?

- Conditional probability:

  $P(a \mid \theta)$

  - Chance of a occurring given θ

- Example θ's:

  - Other event

  - Model parameters

What is the likelihood?

- Informal: same as probability

- Formal: a function of parameter θ that describes the probability of observing data x^n given θ.

- Definition: $L(\theta) \equiv L(\theta; x^n) = P(x^n \,|\, \theta)$

- Usually refers to past events

Sampling

- Definition: $a \sim P$
  - Produce events a following $P(a)$
- Sampling bias
  - Empirical probability of samples != $P(a)$

Expectation

- Definition:
  - $\mathbb{E}_{a \sim P}[f(a)]$
- For any function f
  - $\sum_{a} P(a) f(a)$
  - $\frac{1}{N} \sum_{a \sim P} f(a)$

Linearity of expectation

- $\mathbb{E}_{a\sim P}[\alpha f(a)] = \alpha \mathbb{E}_{a\sim P}[f(a)]$

- $\mathbb{E}_{a\sim P}[f(a) + g(a)] = \mathbb{E}_{a\sim P}[f(a)] + \mathbb{E}_{a\sim P}[g(a)]$

- $\mathbb{E}_{a\sim P}[f(a)g(a)] \neq \mathbb{E}_{a\sim P}[f(a)]\mathbb{E}_{a\sim P}[g(a)]$

Summary

- Event: $a \in [0, \ldots, n-1]$
- Distribution: $P : a \to [0,1]$
- Probability: $P(a)$
- Sampling: $a \sim P$
- Expectation: $\mathbb{E}_{a \sim P}[f(a)]$

[Introduction to Probability, Bertsekas and Tsitsiklis 2002]
[All of Statistics, Wasserman 2004]

Tensors

Overview

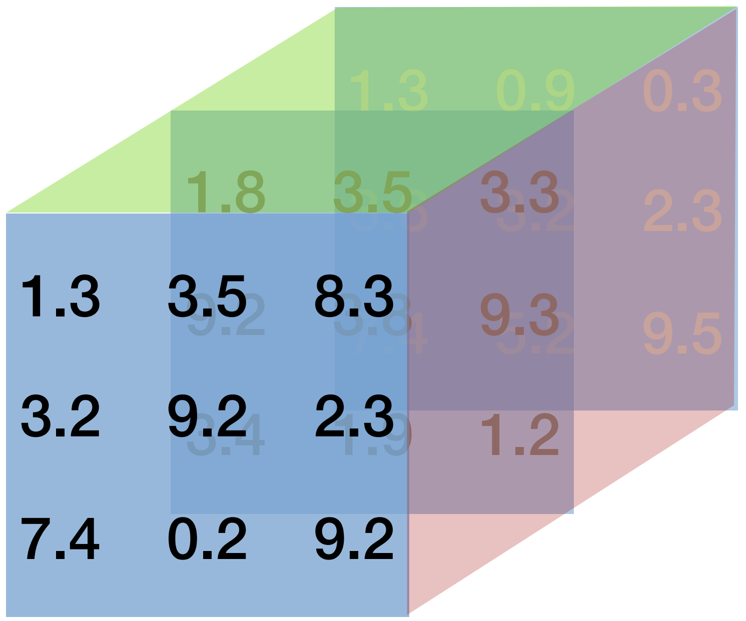
- Tensors: order-d matrices

Tensors

- Notation: $\mathbf{T}$
  - Bold upper case
- Size: $\mathrm{size}(\mathbf{T}) = s_1 \times s_2 \times \ldots \times s_d$
- Order: $\dim(\mathbf{T}) = d$
- Indexing: $\mathbf{T}_{ij\ldots k}$

1.2	3.2	5.8

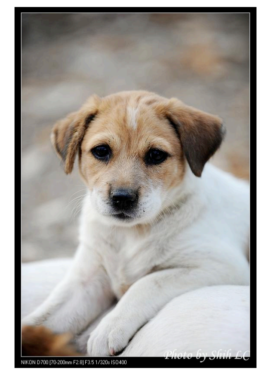
1.3	3.5	8.3
3.2	9.2	2.3
7.4	0.2	9.2

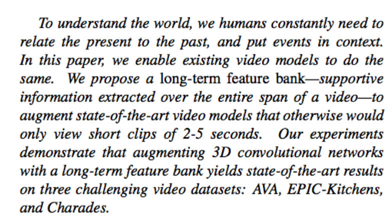


What are Tensors used for?

- In deep learning: everything
- Data
- Parameters
- Intermediate representations
- Input and output of almost any deep network operation

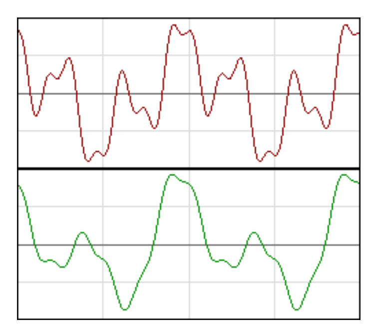
W

H  HxWx3 tensor


To understand the world, we humans constantly need to relate the present to the past, and put events in context. In this paper, we enable existing video models to do the same. We propose a long-term feature bank—supportive information extracted over the entire span of a video—to augment state-of-the-art video models that otherwise would only view short clips of 2-5 seconds. Our experiments demonstrate that augmenting 3D convolutional networks with a long-term feature bank yields state-of-the-art results on three challenging video datasets: AVA, EPIC-Kitchens, and Charades.

N tensor

N-char. doc



TxM tensor

T sec. of audio

Summary

- Tensors: order-d matrices
- Basic building block of deep networks

04

January 23, 2024

Prerequisites: * python 3.6 or newer (e.g. from here http://conda.io) * pytorch and torchvision: https://pytorch.org/ * Pillow: https://pillow.readthedocs.io/en/stable/ * ipython and notebook * numpy, scipy and matplotlib

[1]:
```
import torch
from PIL import Image
import numpy as np
```

[2]:
```
torch.zeros(10)
```

[2]:
```
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[3]:
```
torch.ones(10)
```

[3]:
```
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[4]:
```
torch.ones([4,5])
```

[4]:
```
tensor([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])
```

[5]:
```
v = torch.ones(5)
print( v.dtype, v.shape )
```

```
torch.float32 torch.Size([5])
```

[6]:
```
v = torch.arange(100)
v
```

[6]:
```
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
        54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
        72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
        90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

[7]:
```
v.shape
```

[7]:
```
torch.Size([100])
```

[8]:
```
m = v.view((10,10))
m
```

[8]:
```
tensor([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[9]:
```
m.shape
```

[9]:
```
torch.Size([10, 10])
```

[10]:
```
I = Image.open('cat.jpg')
I
```

[10]:

```
[11]: np.array(I)
```

```
[11]: array([[[194, 206, 220],
          [194, 206, 220],
          [194, 206, 220],
          …,
          [159, 181, 205],
          [156, 180, 204],
          [155, 179, 203]],

         [[195, 207, 221],
          [195, 207, 221],
          [195, 207, 221],
          …,
          [159, 181, 205],
          [156, 180, 204],
          [155, 179, 203]],

         [[196, 208, 222],
          [196, 208, 222],
          [195, 207, 221],
          …,
          [159, 181, 205],
          [156, 180, 204],
          [155, 179, 203]],

         …,

         [[236, 239, 244],
          [236, 239, 244],
          [236, 239, 244],
          …,
          [226, 234, 237],
          [225, 233, 236],
          [224, 232, 235]],

         [[236, 239, 244],
          [236, 239, 244],
          [236, 239, 244],
          …,
          [224, 232, 235],
          [225, 233, 236],
          [224, 232, 235]],

         [[236, 239, 244],
          [236, 239, 244],
```

```
          [236, 239, 244],
          …,
          [219, 227, 230],
          [220, 228, 231],
          [219, 227, 230]]], dtype=uint8)
```

[12]:
```
np.array(I).shape
```

[12]:
```
(853, 1280, 3)
```

[13]:
```
from torchvision import transforms
image_to_tensor = transforms.ToTensor()
image_tensor = image_to_tensor(I)
image_tensor
```

[13]:
```
tensor([[[0.7608, 0.7608, 0.7608,  …, 0.6235, 0.6118, 0.6078],
         [0.7647, 0.7647, 0.7647,  …, 0.6235, 0.6118, 0.6078],
         [0.7686, 0.7686, 0.7647,  …, 0.6235, 0.6118, 0.6078],
         …,
         [0.9255, 0.9255, 0.9255,  …, 0.8863, 0.8824, 0.8784],
         [0.9255, 0.9255, 0.9255,  …, 0.8784, 0.8824, 0.8784],
         [0.9255, 0.9255, 0.9255,  …, 0.8588, 0.8627, 0.8588]],

        [[0.8078, 0.8078, 0.8078,  …, 0.7098, 0.7059, 0.7020],
         [0.8118, 0.8118, 0.8118,  …, 0.7098, 0.7059, 0.7020],
         [0.8157, 0.8157, 0.8118,  …, 0.7098, 0.7059, 0.7020],
         …,
         [0.9373, 0.9373, 0.9373,  …, 0.9176, 0.9137, 0.9098],
         [0.9373, 0.9373, 0.9373,  …, 0.9098, 0.9137, 0.9098],
         [0.9373, 0.9373, 0.9373,  …, 0.8902, 0.8941, 0.8902]],

        [[0.8627, 0.8627, 0.8627,  …, 0.8039, 0.8000, 0.7961],
         [0.8667, 0.8667, 0.8667,  …, 0.8039, 0.8000, 0.7961],
         [0.8706, 0.8706, 0.8667,  …, 0.8039, 0.8000, 0.7961],
         …,
         [0.9569, 0.9569, 0.9569,  …, 0.9294, 0.9255, 0.9216],
         [0.9569, 0.9569, 0.9569,  …, 0.9216, 0.9255, 0.9216],
         [0.9569, 0.9569, 0.9569,  …, 0.9020, 0.9059, 0.9020]]])
```

[14]:
```
image_tensor.shape
```

[14]:
```
torch.Size([3, 853, 1280])
```

[15]:
```
tensor_to_image = transforms.ToPILImage()
tensor_to_image(image_tensor)
```

[15]:

```
[ ]:
```

05

January 23, 2024

[1]:
```
import torch
```

[2]:
```
a = torch.arange(10)
b = torch.arange(10)*10
print( a, b )
```

```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) tensor([ 0, 10, 20, 30, 40, 50, 60, 70,
80, 90])
```

[3]:
```
a+b
```

[3]:
```
tensor([ 0, 11, 22, 33, 44, 55, 66, 77, 88, 99])
```

[4]:
```
a.shape
```

[4]:
```
torch.Size([10])
```

[5]:
```
a[None].shape
```

[5]:
```
torch.Size([1, 10])
```

[6]:
```
a[:,None].shape
```

[6]:
```
torch.Size([10, 1])
```

[7]:
```
c = a[:,None]
c.shape
```

[7]:
```
torch.Size([10, 1])
```

[8]:
```
c[:,0]
```

[8]:
```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[9]:
```
d = a[None]
d.shape
```

[9]:
```
torch.Size([1, 10])
```

[10]:
```
d[0]
```

[10]:
```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[11]:
```
d[0,:]
```

[11]:
```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[12]:
```
torch.ones([3,2]) + 10
```

[12]:
```
tensor([[11., 11.],
        [11., 11.],
        [11., 11.]])
```

[13]:
```
torch.ones([3,2]) + torch.ones([2,3])
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[13], line 1
----> 1 torch.ones([3,2]) + torch.ones([2,3])

RuntimeError: The size of tensor a (2) must match the size of tensor b (3) at␣
↪non-singleton dimension 1
```

[14]:
```
torch.ones([3,1]) + torch.ones([1,3])
```

[14]:
```
tensor([[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]])
```

[15]:
```
a[None,:] + b[:,None]
```

[15]:
```
tensor([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[16]:
```
a[None,:].repeat(10,1)
```

[16]:
```
tensor([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

[17]:
```
b[:,None].repeat(1,10)
```

[17]:
```
tensor([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [10, 10, 10, 10, 10, 10, 10, 10, 10, 10],
        [20, 20, 20, 20, 20, 20, 20, 20, 20, 20],
        [30, 30, 30, 30, 30, 30, 30, 30, 30, 30],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40],
        [50, 50, 50, 50, 50, 50, 50, 50, 50, 50],
        [60, 60, 60, 60, 60, 60, 60, 60, 60, 60],
        [70, 70, 70, 70, 70, 70, 70, 70, 70, 70],
        [80, 80, 80, 80, 80, 80, 80, 80, 80, 80],
        [90, 90, 90, 90, 90, 90, 90, 90, 90, 90]])
```

[18]:
```
a[None,:].repeat(10,1) + b[:,None].repeat(1,10)
```

[18]:
```
tensor([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[19]:
```
a[None,:] * b[:,None]
```

[19]:
```
tensor([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,  10,  20,  30,  40,  50,  60,  70,  80,  90],
        [  0,  20,  40,  60,  80, 100, 120, 140, 160, 180],
        [  0,  30,  60,  90, 120, 150, 180, 210, 240, 270],
        [  0,  40,  80, 120, 160, 200, 240, 280, 320, 360],
        [  0,  50, 100, 150, 200, 250, 300, 350, 400, 450],
        [  0,  60, 120, 180, 240, 300, 360, 420, 480, 540],
        [  0,  70, 140, 210, 280, 350, 420, 490, 560, 630],
```

```
[  0,  80, 160, 240, 320, 400, 480, 560, 640, 720],
[  0,  90, 180, 270, 360, 450, 540, 630, 720, 810]])
```

```
[ ]:
```

06

January 23, 2024

[1]:
```
%pylab inline
import torch

# Load tensorboard to monitor traning (pip install -U tb-nightly)
%load_ext tensorboard
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
import torch.utils.tensorboard as tb
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[3]:
```
logger = tb.SummaryWriter(log_dir+'/test3', flush_secs=1)
logger.add_scalar('first/some_number', 0, global_step=1)
logger.add_scalar('first/some_number', 1, global_step=2)
```

[4]:
```
logger.add_histogram('plots/hist1', np.array([10,2,5,4,2]), global_step=1)
logger.add_histogram('plots/hist2', np.random.rand(20), global_step=1)
```

[5]:
```
logger.add_image('img/1', (np.random.rand(3,100,100)*255).astype(np.uint8),␣
 ↪global_step=1)
```

[6]:
```
logger.add_histogram('plots/hist2', np.random.rand(20), global_step=2)
```

[7]:
```
logger.add_image('img/1', (np.random.rand(3,100,100)*255).astype(np.uint8),␣
 ↪global_step=2)
```

[8]:
```
for i, x in enumerate(np.random.rand(100)):
    logger.add_scalar('first/noise', x+0.1*i, global_step=i)
```

[ ]:
```

```

Summary

Linear algebra

- Vectors & matrices
  - operators
  - gradients & Jacobians

1.2	3.2	5.8

1.3	3.5	8.3
3.2	9.2	2.3
7.4	0.2	9.2

[The Matrix Cookbook, Petersen and Pedersen 2012]

Probabilities

- Event: $a \in [0, \ldots, n-1]$
- Probability: $P(a)$
- Sampling: $a \sim P$
- Expectation: $\mathbb{E}_{a \sim P}[f(a)]$

[Introduction to Probability, Bertsekas and Tsitsiklis 2002]
[All of Statistics, Wasserman 2004]

Tensors

- Tensors: order-d matrices
- Basic building block of deep networks