EngSci 721

Inverse Problems and Learning From Data

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

1. *Basic concepts* [5 lectures + 1 Tutorial]
   Forward vs inverse problems. Well-posed vs ill-posed problems. Algebra and calculus of inverse problems (left and right inverses, generalised and pseudo inverses, resolution operators, matrix calculus). Representing higher dimensional problems (image data etc).
2. *Instability and regularisation in linear and nonlinear problems* [6 lectures + 1 Tutorial]
   Instability and related issues for generalised inverses. Introduction to regularisation and trade-offs. Tikhonov regularisation. Higher-order Tikhonov regularisation. Sparsity and regularisation using different norms. Truncated singular value decomposition. Iterative regularisation, including stochastic/mini-batch gradient descent.
3. *Further topics* [3 lectures + 1 Tutorial]
   Regularisation parameter choice, including statistical and machine learning views of regularisation. Confidence sets for linear and nonlinear models. Physics-informed machine learning and neural networks.

Module overview

Inverse Problems and Learning From Data (*Oliver Maclaren*)
[~14 lectures/3 tutorials]

Lecture 4: Linearity and structure

Topics:

- Matrix multiplication revisited
- Linear spaces and structuring data
  - Vectors and matrices revisited
  - Reshaping with vec and devec
  - Simple example: Image data
  - Kronecker product

Eng Sci 721 : Lecture 4

Linearity & structure

- Matrix multiplication revisited
- Linear spaces & structuring data
  - vectors & matrices revisited
  - Reshaping: vec & devec
  - Example: Images as 2D data
  - Kronecker products

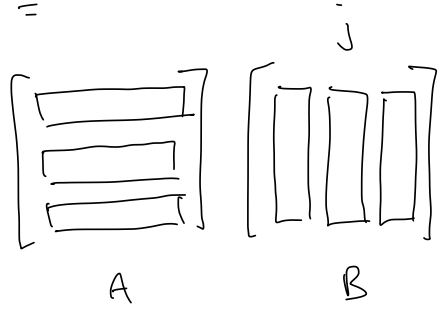
Prelude: Matrix multiplication revisited

It will be helpful to be familiar with multiple ways of thinking of matrix multipl.

Recall the usual 'dot (inner) product' way of thinking about matrix mult.

→ partition into rows & cols as follows:

$AB =$



$$(AB)_{ij} = \text{row}_i(A) \cdot \text{col}_j(B)$$

where $\text{row}_i(A) = A[i,:]$

$\text{col}_j(B) = B[:,j]$

etc.

note: $\text{row}_i(A) = \text{col}_i(A^T)$ etc

[Note:

we use $\text{row}_i(A)$ to represent the ith row of A as a row vector:

$A = \begin{matrix} i \\ \end{matrix}\begin{bmatrix} \quad \end{bmatrix} \rightarrow \text{row}_i(A)$

It's also common to denote this by eg $r_i^T(A)$, or r_i^T if A is clear, which implies we think of the rows we extract as column vectors once extracted, ie if $r_i^T = \text{row}_i(A)$ then r_i is a col vector!

ie $A = \begin{matrix} i \\ \end{matrix}\begin{bmatrix} \quad \end{bmatrix} \xrightarrow{r_i} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

so $(A)_{ij} = \text{row}_i(A)\,\text{col}_j(A)$

$= r_i^T(A)\, C_j(A) = r_i^T C_j$

ie We will use $\boxed{\text{row}_i(A) \;\&\; r_i^T}$ for row vectors & $\boxed{r_i}$ for col vector version of row of A.

]

It turns out that we can also represent this in terms of 'partial' partitions:

eg

(1) $AB = \begin{bmatrix} \quad A \quad \end{bmatrix} \begin{bmatrix} \quad \end{bmatrix}$

$= \begin{bmatrix} A & A & A \end{bmatrix}$ } Note: Block matrix

$\sim \text{col}_1(AB) \quad \sim \text{col}_2(AB) \ldots$ [exercise: prove/verify]

ie $\boxed{\text{col}_j(AB) = A\,\text{col}_j(B)}$

(2) & $AB = \begin{bmatrix} \\ \\ \\ \end{bmatrix} \begin{bmatrix} \quad B \quad \end{bmatrix}$

$= \begin{bmatrix} B \\ B \\ B \end{bmatrix}$ } $\text{row}_1(AB)$ } $\text{row}_2(AB)$

ie $\boxed{\text{row}_i(AB) = \text{row}_i(A)\,B}$

$(= r_i^T B)$

Yet another equivalent way to think of matrix multiplication is in terms of outer products where we partition A & B the other way:

eg

(3)

$$AB = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} \end{bmatrix} \begin{bmatrix} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \end{bmatrix}$$

(A) (B)

$$= \begin{bmatrix} \textcircled{1} \end{bmatrix} \begin{bmatrix} \textcircled{1} \end{bmatrix} + \begin{bmatrix} \textcircled{2} \end{bmatrix} \begin{bmatrix} \textcircled{2} \end{bmatrix} + \begin{bmatrix} \textcircled{3} \end{bmatrix} \begin{bmatrix} \textcircled{3} \end{bmatrix}$$

etc.

ie $$AB = \sum_{i=1}^{n} \text{col}_i(A)\,\text{row}_i(B)$$, n = num cols A / rows B.

$$\left(= \sum_i c_i r_i^T \right)$$

Exercise: prove/convince yourself its true! :)

→

Outer product?

if x, y col vectors, then

$$xy^T = \begin{bmatrix} 1 \\ \vdots \\ m \end{bmatrix} \begin{bmatrix} 1 & \cdots & n \end{bmatrix}$$

'layout' follows y for cols, x for rows

$$= \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & & & \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix}$$

= outer product of x & y

(not x & y^T !)

[we will soon see this is a type of tensor product, $x \otimes_{ten} y = xy^T$

This gives $AB = \sum \text{col}_i(A) \otimes_{ten} \text{col}_i(B^T)$

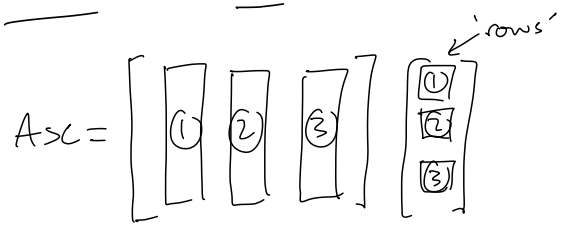
$= \sum c_i(A) \otimes_{ten} r_i(B)$

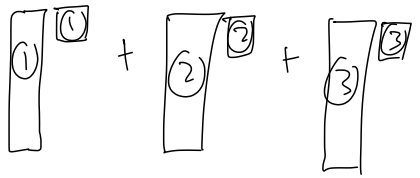
]

Linear combinations?

The outer product point of view specialises for matrix-vector multiplication to:

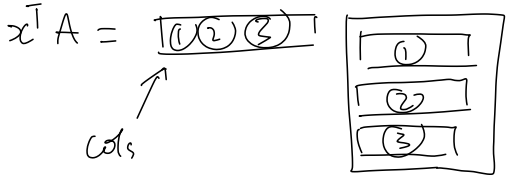
- Matrix time col vector:





linear combo of A's cols, weights = x entries

- Row vector times matrix:



linear combo of A's rows, weights = x entries

Exercise

- Carry out matrix multiplication:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 0 \\ 4 & 1 \end{bmatrix}$$

in

- dot product form
- 'partial' row/col partition forms
- outer product form

& verify the answers are the same

Why? Why all this effort on matrices?

→ Can only handle linear operations on vectors ??

Yes! But
- linearity useful for various approx.
- nonlinear low dim. often ≡ linear high dim
- many things can be thought of as 'vectors'

Linear (vector) spaces

We are used to thinking of vectors as eg 'columns' of numbers

However, in the general, abstract sense, 'vectors' are just 'objects' that we can do 'linear' things with like 'add' two of them together & multiply by 'numbers'

This leads to the axioms of a 'vector space' or 'linear space'

→ see supplement for these axioms

The key point is eg for 'vectors' v_1, v_2 in a 'vector (linear) space' we can form eg $a_1 v_1 + a_2 v_2$ for 'numbers' (scalars) a_1, a_2, & it makes a new vector

Linear operators

- A linear operator maps vectors to vectors & satisfies

$$L(av_1 + bv_2) = aL(v_1) + bL(v_2)$$

for all vectors v_1, v_2 & scalars a, b

- Like a vector space, this is an abstract concept

→ However, in finite dimensions we can represent these by matrices

[See tutorial. Sketch:

$$v = a_1 e_1 + a_2 e_2 + a_3 e_3 + \cdots$$

for scalars a_i & $e_i = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \leftarrow i$

$$L(v) = a_1 L(e_1) + a_2 L(e_2) + \cdots$$

$$= \begin{bmatrix} \vdots & \vdots & \\ L(e_1) & L(e_2) & \cdots \\ \vdots & \vdots & \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \end{bmatrix}$$

]

Matrices as vectors?

Although we often think of an m-dim column vector of numbers as also being an $m \times 1$ matrix, we can also think of matrices as vectors in the sense of being things we can add together & multiply by numbers:

Eg

$$3\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} + 2\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 6 \\ 0 & 3 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ 2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 6 \\ 2 & 3 \end{bmatrix} = "a_1 m_1 + a_2 m_2"$$

for 'vectors' m_1, m_2

Why?

- Data often comes as two or higher-dimensional arrays
  ↳ eg discretised images, videos, temperature fields etc

- More abstractly, it can be also useful to think of linear functions as things that can added, multiplied by numbers etc.

→ We will briefly look at image data soon...

Matrices as vectors cont'd

consider a 2×3 matrix:

$$A = [a_{ij}]_{i=1:2,\, j=1:3}$$ } array entry notation

We clearly have

$$A = a_{11}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + a_{21}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + a_{12}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \dots$$

Hence we have the basis 'vectors'

$$\left\{ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \dots \right\}$$

for the 'vector space' of 2×3 matrices

we write this as $\mathbb{R}^{2\times 3}$

→

Matrices as vectors cont'd

We see that the 'coordinates' of A relative to this 'standard basis' is just a list of numbers:

$$(a_{11}, a_{21}, a_{12}, a_{22}, \dots)$$

Clearly we can think of these as coordinate vectors!

$$A \longleftrightarrow \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ \vdots \\ \vdots \end{bmatrix}$$

We can in fact carry out operations in terms of these 'vectors' & then map back to A if needed ... this is formalised via ... →

vec

The 'vec' operation (also called 'flatten' or 'stack' etc) is defined on matrices as follows.

Given the $m \times n$ matrix:

$$A = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \cdots & \textcircled{n} \end{bmatrix} \Big\} \; m \times n \text{ matrix}$$

Then

$$\text{vec}(A) = \begin{bmatrix} \textcircled{1} \\ \textcircled{2} \\ \vdots \\ \textcircled{n} \end{bmatrix} \Big\} \; mn \times 1 \text{ vector}$$

Note: we are usl the most common 'columnwise stacking' convention here. Others eg rowwise exist.

vec cont'd

Recall the example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$= a_{11}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + a_{21}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + \dots$$

Then $\text{vec}(A) = \begin{pmatrix} \boxed{a_{11}\\ a_{21}} \\ \boxed{a_{12}\\ a_{22}} \\ \boxed{a_{13}\\ a_{23}} \end{pmatrix}$

$$= a_{11}\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + a_{21}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + a_{12}\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \dots$$

$$= a_{11}\,\text{vec}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right) + a_{21}\,\text{vec}\left(\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right) + \dots$$

ie vec { is linear
maps matrix basis vectors to column basis vectors

[Note: can do the same with different basis vectors ie $\text{vec}(A) = \sum_i \alpha_i \text{vec}(B_i)$]

devec

To go back to the array/matrix we 'devec' (unstack, reshape...) the vector.

The shape of the matrix is given by the domain, ie

$$\text{devec} : \mathbb{R}^{mn} \to \mathbb{R}^{m \times n}$$

We can also just eg indicate the number of columns (say)

devec_n (mn vector) $\to$ m x n matrix

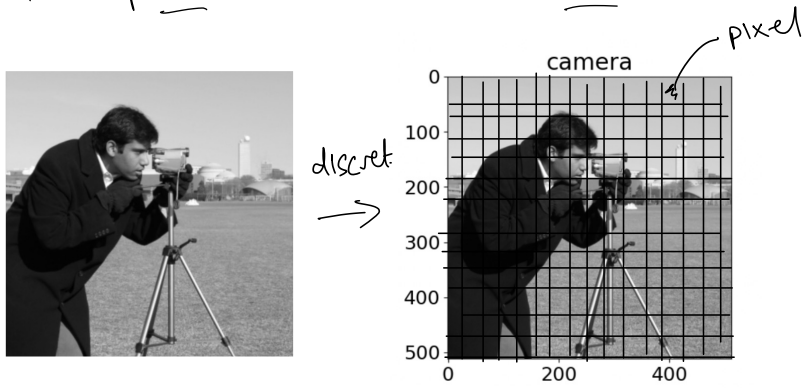
Eg

$$\text{devec}_3\left(\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}\right) = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

[we assume the same columnwise stacking/unstacking convention]

Why? Images pt. 1

- A (greyscale) image can be discretised into pixels & stored in a matrix:



$X(i,j) \in [0,1]$

(colour images can be handled using the same ideas to be discussed...)

- However, many (though not all!) machine learning, statistics, linear algebra etc algorithms & computational libraries require | vector | 'features', 'covariates', 'inputs' etc

Image data for vector input alg.

→ can just use | vec | ! (or 'flatten')

(tho some modern ML approaches accept multi-dim arrays, ie 'tensor data', directly → vec still useful conceptually)

Eg in regression & data sci/ML libraries like scikit-learn we arrange our input in a 'design matrix' like:

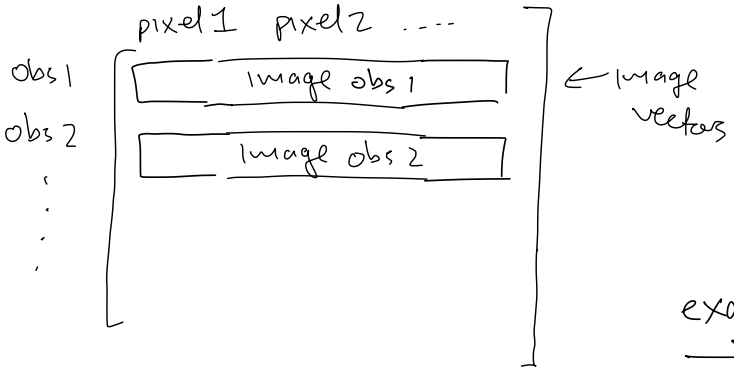
$$D = \begin{pmatrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{pmatrix}$$

feature1 feature2 ...

obs 1
obs 2
obs 3

→

For image data this becomes



example →

etc.

Similary many math. models of the imaging process ('taking a photo') assume a linear mapping:

sharp image → blurred image + noise

assume linear blurring.

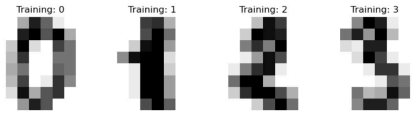
→

scikit-learn example (from docs at scikit-learn.org)

Digits dataset

The digits dataset consists of 8x8 pixel images of digits. The `images` attribute of the dataset stores 8x8 arrays of grayscale values for each image. We will use these arrays to visualize the first 4 images. The `target` attribute of the dataset stores the digit each image represents and this is included in the title of the 4 plots below.

8x8 matrix greyscale values



Classification

To apply a classifier on this data, we need to flatten the images, turning each 2-D array of grayscale values from shape `(8, 8)` into shape `(64,)`. Subsequently, the entire dataset will be of shape `(n_samples, n_features)`, where `n_samples` is the number of images and `n_features` is the total number of pixels in each image.

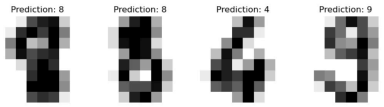
flatten = vec

features = pixels

```
# flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
clf = svm.SVC(gamma=0.001)

# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False
)

# Learn the digits on the train subset
clf.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = clf.predict(X_test)
```

Below we visualize the first 4 test samples and show their predicted digit value in the title.

```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")
```



`classification_report` builds a text report showing the main classification metrics.

Blurring

Sometimes we can write down the model as acting directly on the image matrix (non-vec)

Eg. linear vertical (column) blurring/smoothing

$$A_c X = \left[A_c C_1(X) \;\; A_c C_2(X) \ldots \right]$$

matrix rep. of convol. $\Big\{$ where $A_c = \frac{1}{3}\begin{bmatrix} 1 & 1 & 0 & \cdots & 1 \\ 0 & 1 & 1 & 1 & \ldots \\ 0 & 0 & 1 & 1 & 1 \ldots \end{bmatrix}$ (periodic blur)

In general though, we can represent a linear blur (or linear filter in gen):

$$\boxed{A\,\mathrm{vec}(X) = \mathrm{vec}(B)}$$

where X : original image

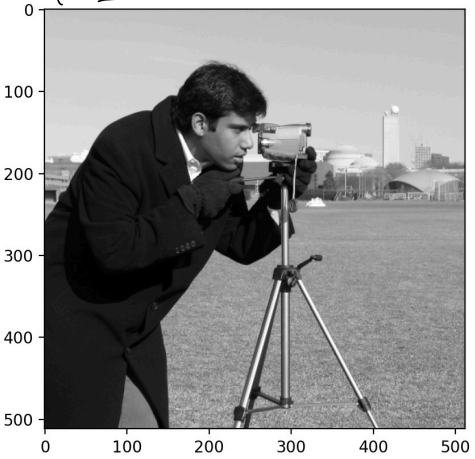
B : blurred image

→ see tut/assignment.

Vertical blur example

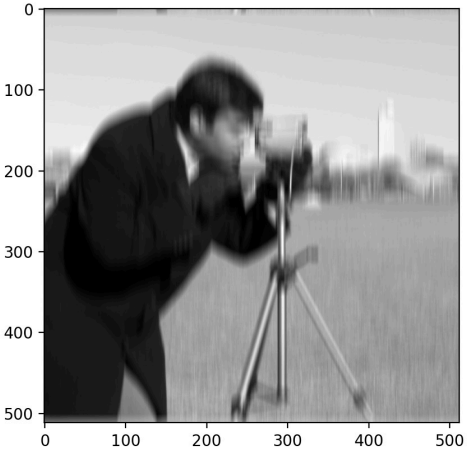
```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt

caller = getattr(data, 'camera')
X = caller()
X = X/np.max(X)

plt.imshow(X,cmap='gray')
plt.show()

Ac = np.zeros(X.shape)
a_row_vec = np.zeros(X.shape[0])
a_row_vec[0:11] = 1/21
a_row_vec[-10:] = 1/21

for i in range(0,len(a_row_vec)):
    Ac[i,:] = np.roll(a_row_vec,i)
print(Ac)

B = np.dot(Ac,X )
plt.imshow(B,cmap='gray')
plt.show()
```

Sharp:



→

Blurred



Kronecker product

To work with 'vec' & hence 'flatten' 'linear' objects to their (coordinate) 'vector' forms, we need one more ingredient: the

Kronecker product : $\otimes_{kron}$

This is like the 'outer' or 'tensor' product we saw before (& is actually a type of 'tensor product' in the general sense) but arranges the results in a way naturally suited to 'vec' & vector-matrix representations

* **Note**: we will use $\otimes_{kron}$ much more than $\otimes_{Tens}$, so will just use $\otimes$ for $\otimes_{kron}$ unless need both

Kronecker product

The outer tensor product of two vectors gives a matrix of all pairs of element products

In contrast, the Kronecker (tensor) product of two vectors gives a vector of all such element products:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$x \otimes y = \begin{bmatrix} \begin{bmatrix} x_1 y_1 \\ x_1 y_2 \\ x_1 y_3 \\ \vdots \\ x_1 y_n \end{bmatrix} \\ \begin{bmatrix} x_2 y_1 \\ x_2 y_2 \\ \vdots \end{bmatrix} \\ \vdots \end{bmatrix} \Bigg\} \text{ vector of length } mn$$

Note:

$$x \otimes_{Kron} y = vec\left(y \otimes_{tens} x\right)$$

$$= vec\left(y x^T\right)$$

ie the Kronecker product 'vectorises' the 'usual' tensor outer product (note order of x, y thro')

Kronecker product cont'd

The Kronecker product is defined for matrices too.

It again keeps things as 'flat as possible'

vector $\otimes$ vector $\rightarrow$ vector

matrix $\otimes$ matrix $\rightarrow$ matrix

matrix $\otimes$ vector $\rightarrow$ matrix

The definition for $A: m \times n$, $B: p \times q$ is given in block form as:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & & & \\ \vdots & & & \\ a_{m1}B & \cdots & & a_{mn}B \end{bmatrix}$$

$\rightarrow$ This is an $mp \times nq$ matrix!

$\rightarrow$ The idea is we 'lay out' the second matrix 'according to' the first: each element of A gets a full copy of B, scaled by that element

Examples

(A) $x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

$$x \otimes y = \begin{bmatrix} x_1 y \\ x_2 y \\ \vdots \\ x_m y \end{bmatrix}, \text{ where:}$$

$$x_1 y = \begin{bmatrix} x_1 y_1 \\ x_1 y_2 \\ x_1 y_3 \\ \vdots \end{bmatrix}, \quad x_2 y = \begin{bmatrix} x_2 y_1 \\ x_2 y_2 \\ x_3 y_2 \\ \vdots \end{bmatrix}$$

etc.

so $x \otimes y$ is mn length vector

$\longrightarrow$

(B) $x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

$$\Rightarrow x \otimes y^T = \begin{bmatrix} x_1 y^T \\ x_2 y^T \\ \vdots \\ x_m y^T \end{bmatrix}$$

$$= \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & \vdots \\ \vdots & \vdots & & \vdots \\ x_m y_1 & \cdots & \cdots & x_m y_n \end{bmatrix}$$

$$= x \otimes_{\text{tens.}} y \; !$$

(c) $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$x^T \otimes I_2 = \begin{bmatrix} x_1 I_2 & x_2 I_2 & x_3 I_2 \end{bmatrix}$

$= \begin{bmatrix} x_1 & 0 & x_2 & 0 & x_3 & 0 \\ 0 & x_1 & 0 & x_2 & 0 & x_3 \end{bmatrix}$

($2 \times 3 \cdot 2 = 2 \times 6$ matrix ✓)

---

vec & $\otimes$

→ Play nice together!

Key relationship for converting matrix eq^{n}s back & forward

* $\boxed{\text{vec}(ABC) = (C^T \otimes I)\,\text{vec}(B)}$ *

[ie: $\overbrace{ABC}$
move C to back, transpose & $\otimes$ with A
B comes to front & is vectorised.]

Note: $\text{vec}(AB) = \text{vec}(IAB)$

$= (B^T \otimes I)\,\text{vec}(A)$

$= \text{vec}(ABI)$

$= (I \otimes A)\,\text{vec}(B)$

etc.

→ Together, vec & $\otimes$ allow us to restructure many problems as 'linear matrix vector' problems

→ sometimes want vec form, others devec.

Some other properties of $\otimes$:

$$(A \otimes B)^T = A^T \otimes B^T$$

$$(A \otimes B)^+ = A^+ \otimes B^+$$

(* where $^+$ denotes pseudo inverse (mul. inv. if exists))

Also :

$$(A \otimes B)(C \otimes D) = AC \otimes BD$$

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$

etc. (* more imp. for us)

---

example

x : $n \times 1$
A : $m \times n$
b : $m \times 1$
} $Ax = b$

- Suppose want to treat A as unknown } solve for A given x & b.

$$vec(Ax) = vec(I_m A x)$$

$$= (x^T \otimes I_m)\, vec(A)$$

$$vec(b) = b = Ax \quad \text{(here only)}$$

eg $m = 2, n = 3$

$$A = {}_2\begin{bmatrix} & & \\ & & \end{bmatrix}_3, \quad x = \begin{bmatrix} \\ \\ \end{bmatrix}_3$$

$$x^T \otimes I_2 = \begin{bmatrix} x_1 & 0 & x_2 & 0 & x_3 & 0 \\ 0 & x_1 & 0 & x_2 & 0 & x_3 \end{bmatrix}$$

$$vec(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \\ a_{13} \\ a_{23} \end{bmatrix}, \quad vec(b)$$

$$\begin{bmatrix} x_1 & 0 & x_2 & 0 & x_3 & 0 \\ 0 & x_1 & 0 & x_2 & 0 & x_3 \end{bmatrix} \begin{bmatrix} \\ vec(A) \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \end{bmatrix}$$

$$= Ax = b \ \checkmark$$

$$\Rightarrow (x^T \otimes I_2)\, \underbrace{vec\, A}_{\text{unknown}} = vec(b)$$ } matrix eq^n for vec(A)

Exercises for $\otimes$ & vec:

- Independent vertical & horizontal blurring of an image matrix can be done by pre & post multiplying the image X:

$$\boxed{A_c X A_r^T}$$

where A_c is a vertical blur & A_r^T is a horizontal blur

→ show this can be represented as a single linear operation on the image vector:

$$\boxed{M\,\mathrm{vec}(X)}$$

for some M you should determine

- Matrices can represent linear functions mapping vectors to vectors. They can also represent nonlinear but bilinear functionals on pairs of vectors eg

$$\boxed{B(x,y) = x^T A y}$$

(see handout for more).

for $A: n \times n$, $x: n \times 1$, $y: n \times 1$

→ write this as a linear dot product in higher dim:

$$\boxed{v^T \mathrm{vec}(A)}$$

for some v you determine.