More Linear Algebra

Alexander G. Ororbia II

COGS-621: Foundations of Scientific Computing

9/25/2025

Elementwise composed functions

- *Can build from simple routines:* *cos(.)*, *sin(.)*, *exp(.)*, etc. (the "." means argument)

Softmax: $\phi(\mathbf{v}) = \frac{\exp(\mathbf{v})}{\sum_{c=1}^{C} \exp(\mathbf{v}_c)}$

Sigmoid: $\phi(\mathbf{v}) = \sigma(\mathbf{v}) = \frac{1}{1+e^{-\mathbf{v}}}$

Let's write these out to our Python interpreter!

NumPy Function
np.cos, np.sin, np.tan
np.arccos, np.arcsin, np.arctan
np.cosh, np.sinh, np.tanh
np.arccosh, np.arcsinh, np.arctanh
np.sqrt
np.exp
np.log, np.log2, np.log10

NumPy Function
np.add, np.subtract, np.multiply, np.divide
np.power
np.remainder
np.reciprocal
np.real, np.imag, np.conj
np.sign, np.abs
np.floor, np.ceil, np.rint
np.round

Transposing/manipulation in NumPy

$(\mathbf{A}^{\mathrm{T}})_{i,j} = A_{j,i}$

$\boldsymbol{x} = [x_1, .., x_n]^{\mathrm{T}}$

***Table 2-12.** Summary of NumPy Functions for Array Operations*

Function	Description
`np.transpose`, `np.ndarray.transpose`, `np.ndarray.T`	Transpose (reverse axes) an array.
`np.fliplr`/`np.flipud`	Reverse the elements in each row/column.
`np.rot90`	Rotate the elements along the first two axes by 90 degrees.
`np.sort`, `np.ndarray.sort`	Sort an array's elements along a specified axis (which defaults to the last axis of the array). The `np.ndarray` method `sort` performs the sorting in place, modifying the input array.

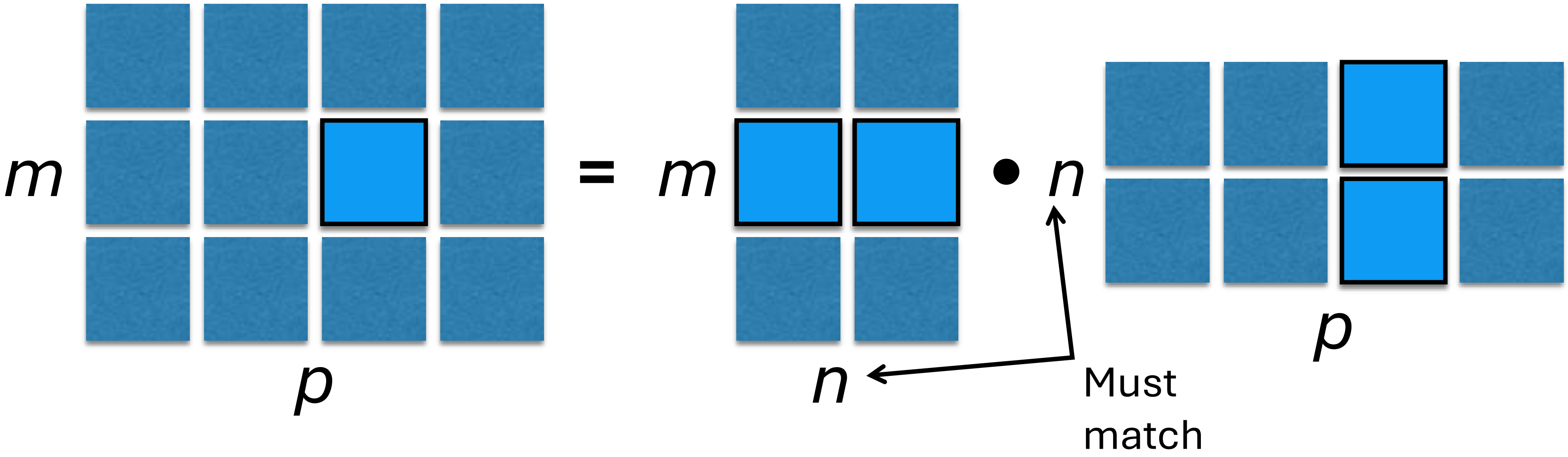
$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Matrix multiplication

- For product $\boldsymbol{C}=\boldsymbol{AB}$ to be defined, $\boldsymbol{A}$ has to have the same no. of columns as the no. of rows of $\boldsymbol{B}$
  - If $\boldsymbol{A}$ is of shape mxn and B is of shape nxp then *matrix product* $\boldsymbol{C}$ is of shape mxp

$$C = AB \Rightarrow C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

- Dot product between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of same dimensionality is the matrix product $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$
- We can think of matrix product $C=AB$ as computing C_{ij} the dot product of row i of A and column j of B



Referred to sometimes as matrix-matrix product or matrix-vector product (or matrix multiply)

Matrix-matrix multiply

- Matrix-Matrix multiply (outer product)
  - Vector-Vector multiply (dot product)
- The usual workhorse of statistical learning
- Vectorizes sums of products (builds on dot product)

0.5	-0.7
-0.69	1.8

*

0.5	-0.7
-0.69	1.8

=

(.5 * .5) + (-.7 * -.69)	(.5 * -.7) + (-.7 * 1.8)
(-.69 * .5) + (1.8 * -.69)	(-.69 * -.7) + (1.8 * 1.8)

Matrix/vector products

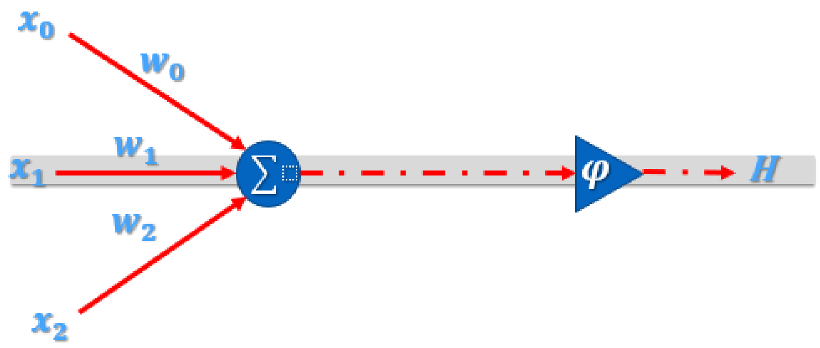
- **Inner (dot) product** - combine 2 vectors into scalar to measure their alignment/similarity (*reduction*)
- **Outer product** - combines 2 vector into matrix to capture pairwise interactions (*expansion*)

Table 2-13. Summary of NumPy Functions for Matrix Operations

NumPy Function	Description
`np.dot`	Matrix multiplication (dot product) between two given arrays representing vectors, arrays, or tensors
`np.inner`	Scalar multiplication (inner product) between two arrays representing vectors
`np.cross`	The cross product between two arrays that represent vectors
`np.tensordot`	Dot product along specified axes of multidimensional arrays
`np.outer`	Outer product (tensor product of vectors) between two arrays representing vectors
`np.kron`	Kronecker product (tensor product of matrices) between arrays representing matrices and higher-dimensional arrays
`np.einsum`	Evaluates Einstein's summation convention for multidimensional arrays

- You can emulate some behaviors with dot product and transpose
  - $Dot(x, y) = x^T \cdot y$; $Outer(x, y) = x \cdot y^T$
  - Can also use np.matmul (@) to get same effect (lines up with other packages)

Vector form (one unit)



This calculates activation value of single (output) unit that is connected to 3 (input) sensors.

$$h_0: \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \varphi(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)$$

Vector form (two units)

This vectorization easily generalizes to multiple (3) sensors feeding into multiple (2) units.

$$\begin{matrix} h_0: \\ h_1: \end{matrix} \begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \varphi(w_0 * x_0 + w_1 * x_1 + w_2 * x_2) \\ \varphi(w_3 * x_0 + w_4 * x_1 + w_5 * x_2) \end{bmatrix}$$

Known as vectorization!

Now let us fully vectorize this!

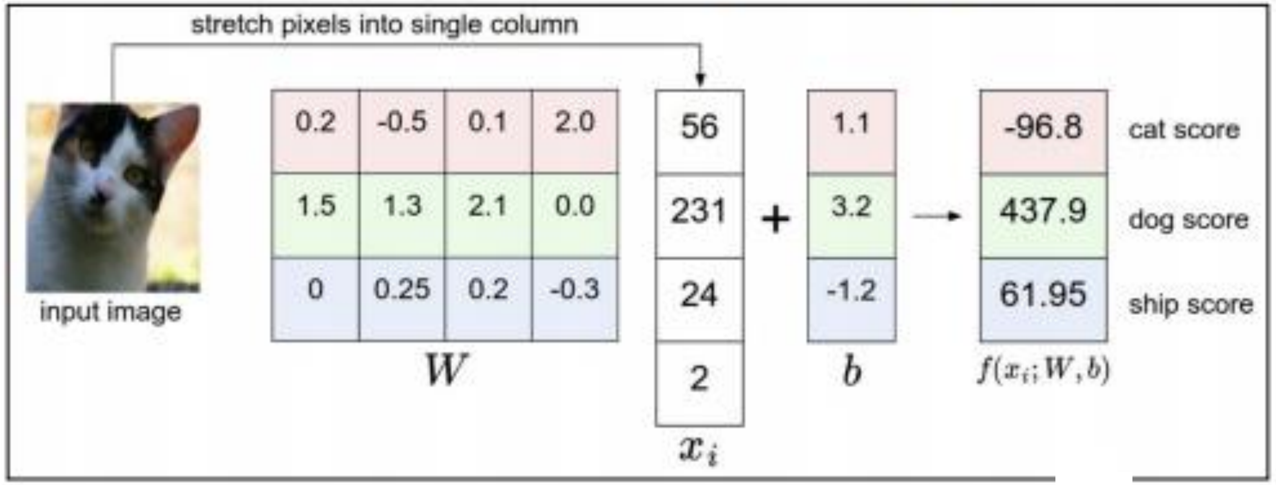
This vectorization is also important for
formulating **mini-batches**.
(Good for GPU-based processing.)

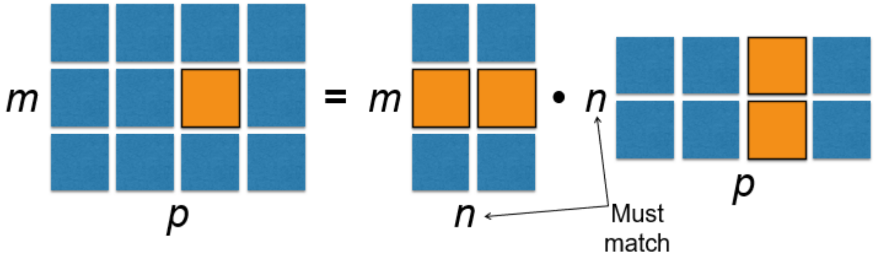
$$
\begin{matrix} h_0: \\ h_1: \end{matrix}
\begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \end{bmatrix}
*
\begin{bmatrix} x_0 & x_3 \\ x_1 & x_4 \\ x_2 & x_5 \end{bmatrix}
=
\begin{bmatrix} \varphi(w_0 * x_0 + \ldots) & \varphi(w_0 * x_3 + \ldots) \\ \varphi(w_3 * x_0 + \ldots) & \varphi(w_3 * x_3 + \ldots) \end{bmatrix}
$$

Tensors in statistical learning

Vector $\boldsymbol{x}$ is converted into vector $\boldsymbol{y}$ by multiplying $\boldsymbol{x}$ by a matrix $\boldsymbol{W}$

A linear classifier $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}^{\mathrm{T}} + \boldsymbol{b}$





Questions?