



Machine Learning: More Elements of *Linear Algebra*

Alexander G. Ororbia II
Introduction to Machine Learning
CSCI-335
8/31/2026

Elementwise Operators (LA)

Table 2-6. Operators for Elementwise Arithmetic Operation on NumPy Arrays

Operator	Operation
<code>+, +=</code>	Addition
<code>-, -=</code>	Subtraction
<code>*, *=</code>	Multiplication
<code>/, /=</code>	Division
<code>//, //=</code>	Integer division
<code>** , **=</code>	Exponentiation

← `.` Hadamard product

- We also developed some other LA building blocks;
 - Tensor/matrix/vector multiply, transpose, etc.

Elementwise (Same-Shape) Mathematics

- Addition “+” (+)

$$C = A+B \Rightarrow C_{i,j} = A_{i,j} + B_{i,j}$$

0.5	-0.7
-0.69	1.8

+

0.5	0.7
-0.69	1.8

=

.5 + .5 = 1.0	-.7 - .7 = -0.0
-.69 - .69 = -1.38	1.8 + 1.8 = 3.6

- Subtraction “-” (-)

- Multiplication (Hadamard product) “*” (⊙)

0.5	-0.7
-0.69	1.8

⊙

0.5	0.7
-0.69	1.8

=

.5 * .5 = .25	-.7 * .7 = -.49
-.69 * -.69 = .4761	1.8 * 1.8 = 3.24

- Division “/” (/)

Vectorization and Broadcasting

Adding a 3x3 matrix to a 1x3 row vector:

11	12	13		1	2	3		12	14	16
21	22	23	+	1	2	3	=	22	24	26
31	32	33		1	2	3		32	34	36

Adding a 3x3 matrix to a 3x1 column vector:

11	12	13		1	1	1		12	13	14
21	22	23	+	2	2	2	=	23	24	25
31	32	33		3	3	3		34	35	36

True elements

Broadcasted elements

- **Vectorization**: store numerical data in arrays to use batch operations (applied to all values in array)

- Avoids explicit loops
- Makes code easier to maintain / more concise, better performance
- Binary op well-defined if two arguments are same shape

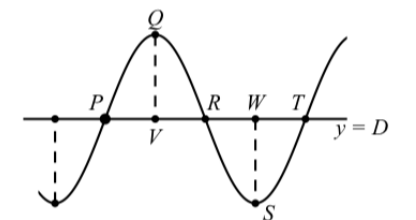
Broadcasting: effective array expansion

- Array + scalar – scalar distributed (& op applied) to each element in array
- Unequal shape arrays – if smaller array can be broadcasted to larger array (**rule**: if array axes on 1-on-1 basis have same length or length of 1)
 - If an array has fewer axes, dummy axes are padded on until dimensions of arrays agree

More Elementwise Functions (Vectorized Operators)

Table 2-7. Selection of NumPy Functions for Elementwise Elementary Mathematical Functions

NumPy Function	Description
<code>np.cos</code> , <code>np.sin</code> , <code>np.tan</code>	Trigonometric functions
<code>np.arccos</code> , <code>np.arcsin</code> , <code>np.arctan</code>	Inverse trigonometric functions
<code>np.cosh</code> , <code>np.sinh</code> , <code>np.tanh</code>	Hyperbolic trigonometric functions
<code>np.arccosh</code> , <code>np.arcsinh</code> , <code>np.arctanh</code>	Inverse hyperbolic trigonometric functions
<code>np.sqrt</code>	Square root
<code>np.exp</code>	Exponential
<code>np.log</code> , <code>np.log2</code> , <code>np.log10</code>	Logarithms of base e, 2, and 10, respectively



Sinusoid function

- NumPy contains many vectorized functions – elementwise evaluation of basic mathematical functions (over tensors)
 - Take in tensor/ndarray, return tensor/ndarray of same shape as input (though not necessarily of same type)

Table 2-8. Summary of NumPy Functions for Elementwise Mathematical Operations

NumPy Function	Description
<code>np.add</code> , <code>np.subtract</code> , <code>np.multiply</code> , <code>np.divide</code>	Addition, subtraction, multiplication, and division of two NumPy arrays
<code>np.power</code>	Raises first input argument to the power of the second input argument (applied elementwise)
<code>np.remainder</code>	The remainder of the division
<code>np.reciprocal</code>	The reciprocal (inverse) of each element
<code>np.real</code> , <code>np.imag</code> , <code>np.conj</code>	The real part, imaginary part, and the complex conjugate of the elements in the input arrays
<code>np.sign</code> , <code>np.abs</code>	The sign and the absolute value
<code>np.floor</code> , <code>np.ceil</code> , <code>np rint</code>	Converts to integer values
<code>np.round</code>	Rounds to a given number of decimals

- Elementwise functions can be single or multi-argument
- Can use the `np.vectorize` to convert some custom functions to those that operate on tensors/ndarrays

NumPy aggregation functions (aggregators)

Table 2-9. NumPy Functions for Calculating Aggregates of NumPy Arrays

NumPy Function	Description
<code>np.sum</code>	The sum of all elements Σ = summation (capital sigma)
<code>np.prod</code>	The product of all elements Π = product (capital pi)
<code>np.min</code> , <code>np.max</code>	The minimum/maximum value in an array
<code>np.argmax</code> , <code>np.argmin</code>	The index of the minimum/maximum value in an array

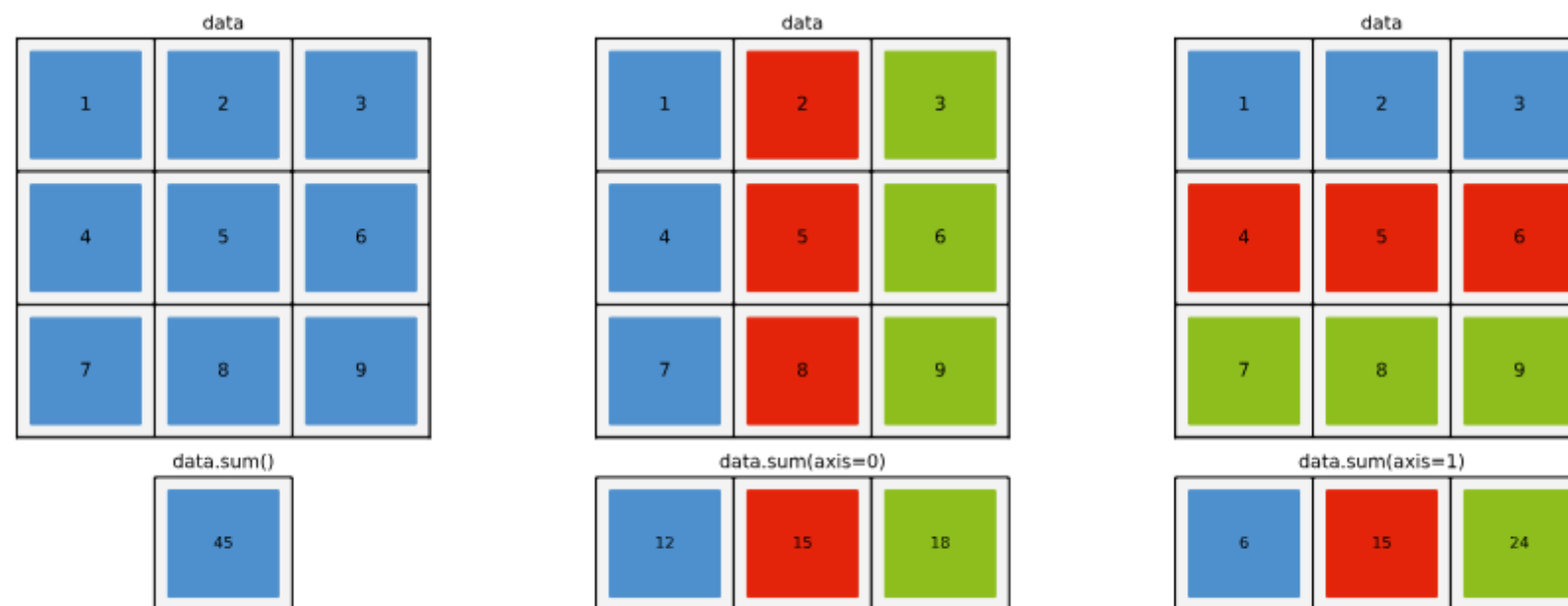


Figure 2-3. Illustration of array aggregation functions along all axes (left), the first axis (center), and the second axis (right) of a two-dimensional array of shape 3×3

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

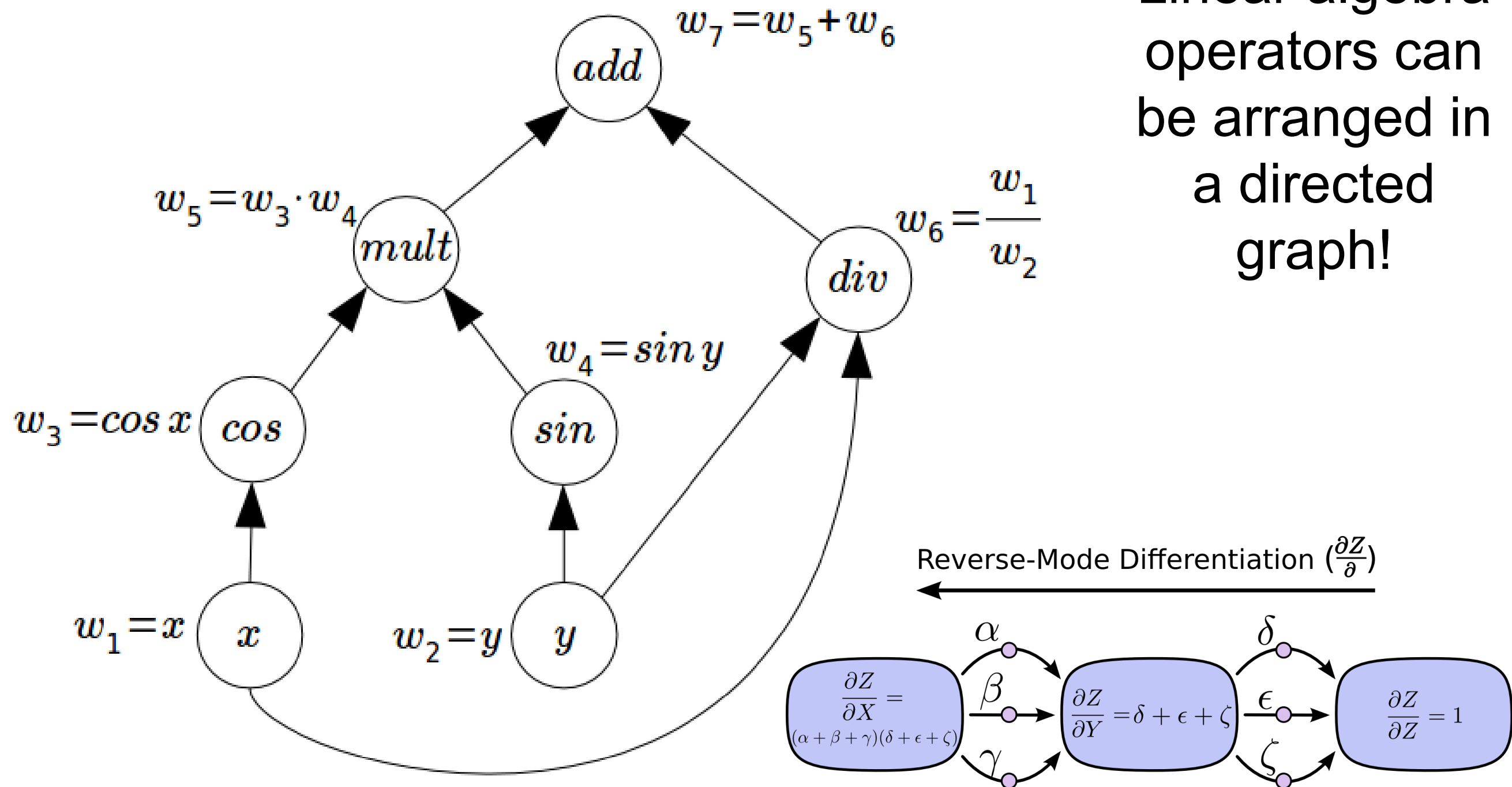
np.remainder
np.reciprocal
np.real, np.imag, np.conj

np.sign, np.abs
np.floor, np.ceil, np rint
np.round

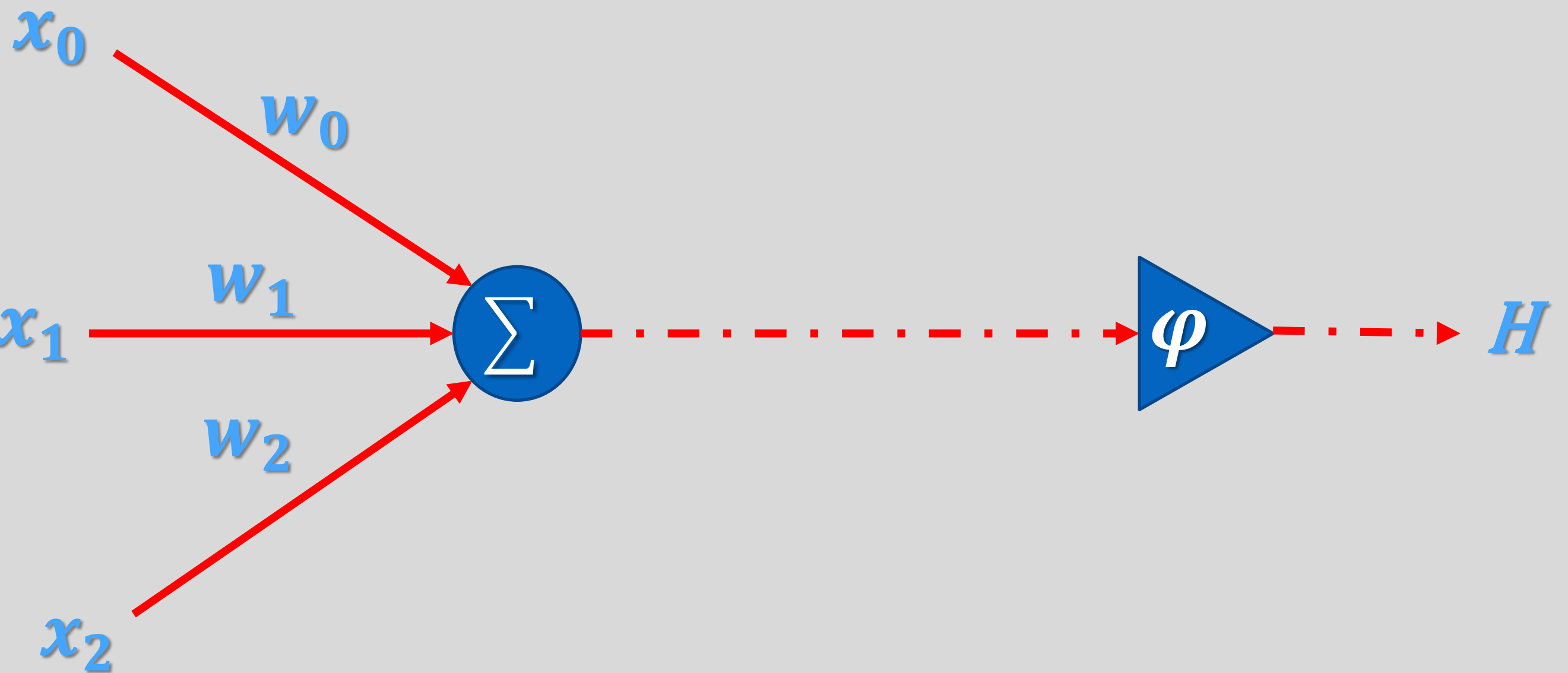
Why Do We Care?

Computation Graphs

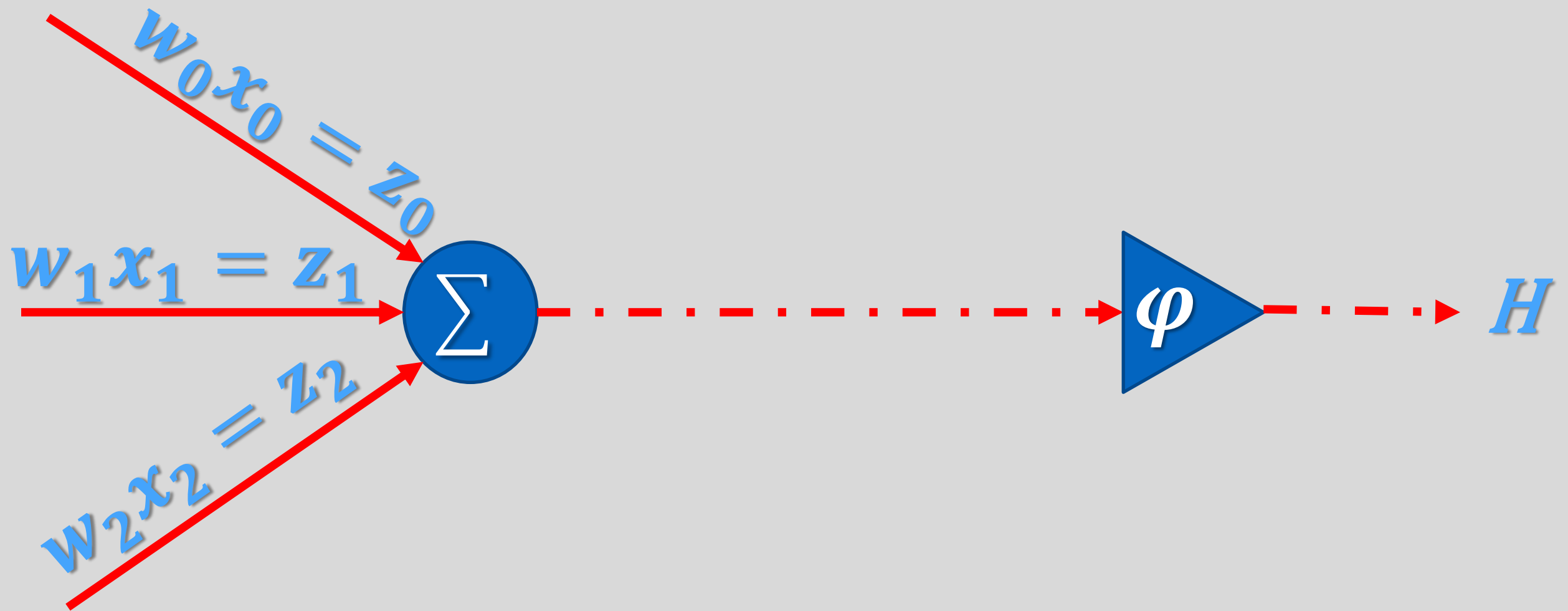
Linear algebra operators can be arranged in a directed graph!



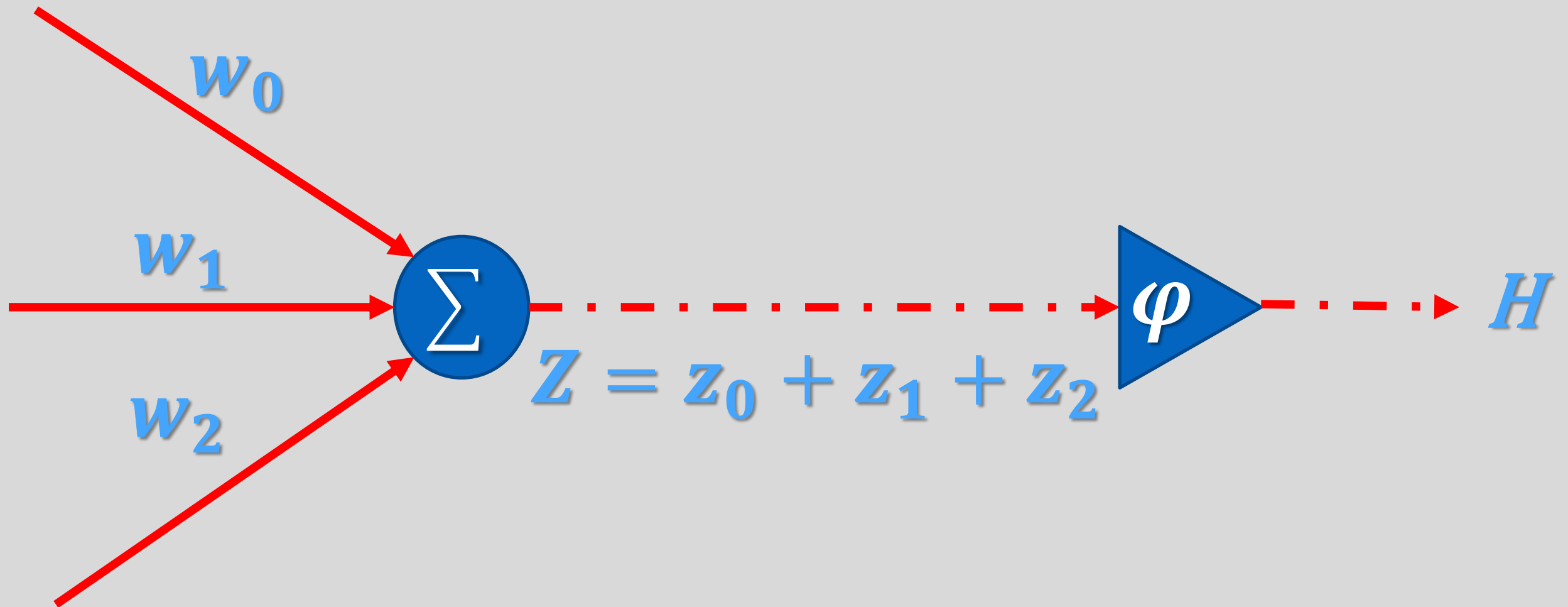
A Simple Linear Predictor



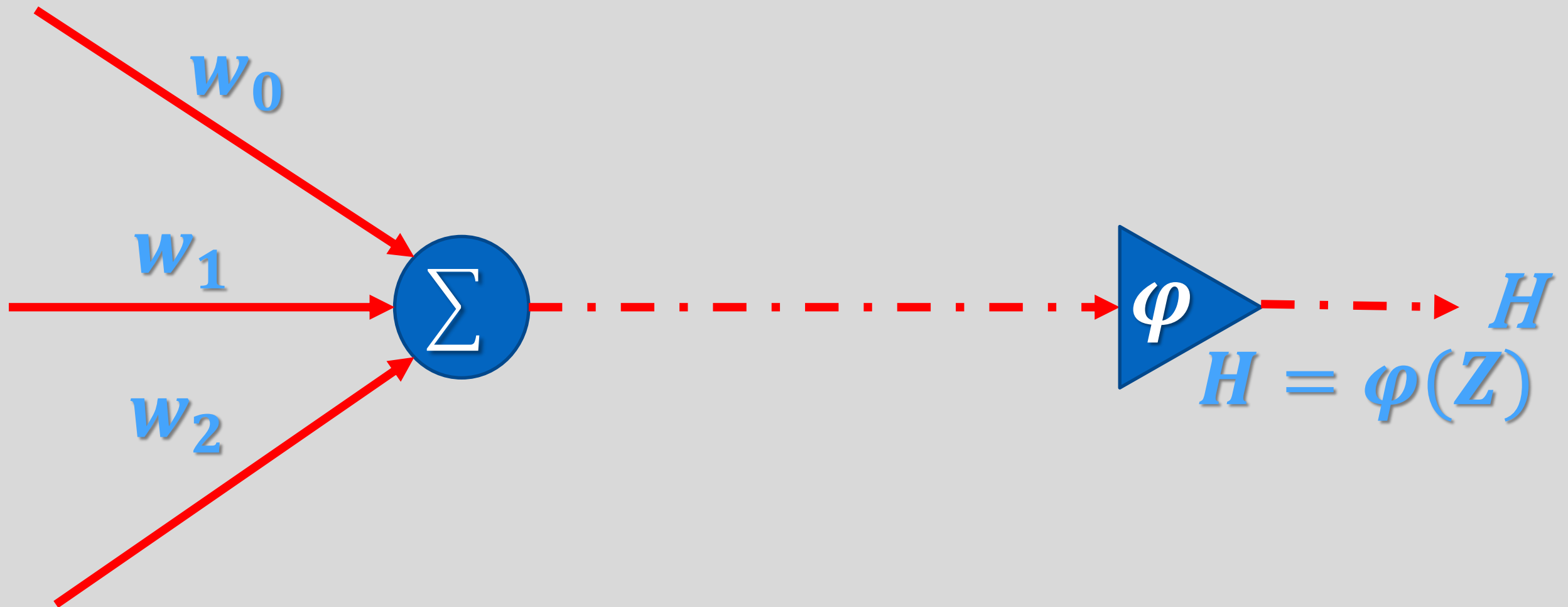
A Simple Linear Predictor



A Simple Linear Predictor



A Simple Linear Predictor



Vector Form (One Unit)

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

$$h_0: \begin{array}{|c|c|c|} \hline w_0 & w_1 & w_2 \\ \hline \end{array} * \begin{array}{|c|} \hline x_0 \\ \hline x_1 \\ \hline x_2 \\ \hline \end{array} = \boxed{\varphi(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)}$$

Vector Form (Two Units)

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

$$\begin{array}{l} h_0: \\ h_1: \end{array} \begin{array}{|c|c|c|} \hline w_0 & w_1 & w_2 \\ \hline w_3 & w_4 & w_5 \\ \hline \end{array} * \begin{array}{|c|} \hline x_0 \\ \hline x_1 \\ \hline x_2 \\ \hline \end{array} = \begin{array}{|c|} \hline \varphi(w_0 * x_0 + w_1 * x_1 + w_2 * x_2) \\ \hline \varphi(w_3 * x_0 + w_4 * x_1 + w_5 * x_2) \\ \hline \end{array}$$

Known as vectorization!

Now Let Us Fully Vectorize This!

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

$$\begin{array}{l} h_0: \\ h_1: \end{array} \begin{array}{|c|c|c|} \hline w_0 & w_1 & w_2 \\ \hline w_3 & w_4 & w_5 \\ \hline \end{array} * \begin{array}{|c|c|} \hline x_0 & x_3 \\ \hline x_1 & x_4 \\ \hline x_2 & x_5 \\ \hline \end{array} = \begin{array}{|c|c|} \hline \varphi(w_0 * x_0 + \dots) & \varphi(w_0 * x_3 + \dots) \\ \hline \varphi(w_3 * x_0 + \dots) & \varphi(w_3 * x_3 + \dots) \\ \hline \end{array}$$

Identity Matrices

- Matrix inversion is a powerful tool to analytically solve $A\mathbf{x}=\mathbf{b}$
- Needs concept of Identity matrix
- Identity matrix does not change value of vector when we multiply the vector by identity matrix

– Denote identity matrix that preserves n-dimensional vectors as I_n

– Formally $I_n \in \mathbb{R}^{n \times n}$ and $\forall \mathbf{x} \in \mathbb{R}^n, I_n \mathbf{x} = \mathbf{x}$

– Example of I_3 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Creating identity matrix in NumPy:

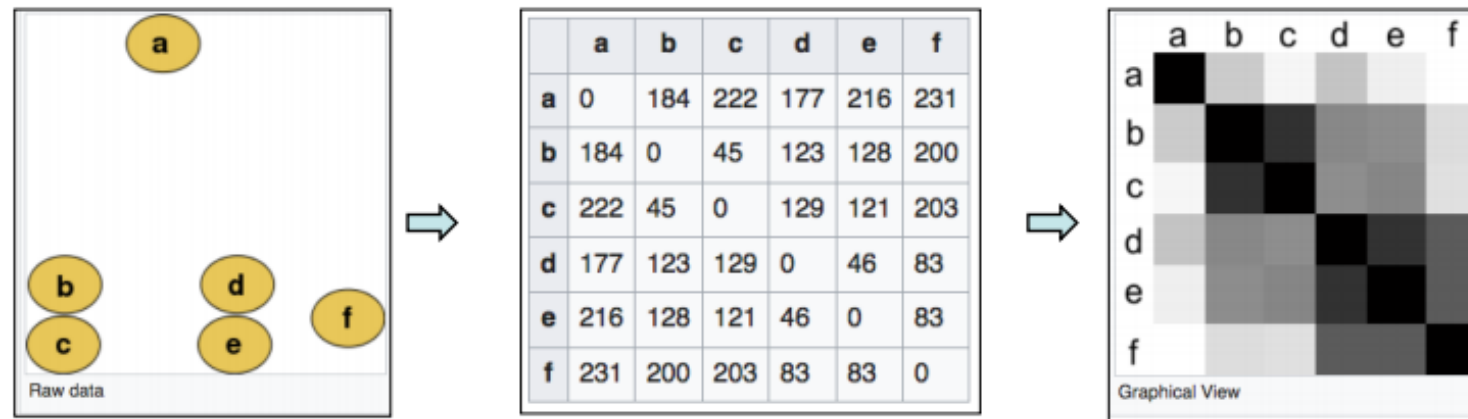
```
In [62]: np.identity(4)
Out[62]: array([[ 1.,  0.,  0.,  0.],
                [ 0.,  1.,  0.,  0.],
                [ 0.,  0.,  1.,  0.],
                [ 0.,  0.,  0.,  1.]])
```

```
In [63]: np.eye(3, k=1)
Out[63]: array([[ 0.,  1.,  0.],
                [ 0.,  0.,  1.],
                [ 0.,  0.,  0.]])
```

```
In [64]: np.eye(3, k=-1)
Out[64]: array([[ 0.,  0.,  0.],
                [ 1.,  0.,  0.],
                [ 0.,  1.,  0.]])
```

The Symmetric Matrix

- A symmetric matrix equals its transpose: $A = A^T$
 - E.g., a distance matrix is symmetric with $A_{ij} = A_{ji}$



- E.g., covariance matrices are symmetric

$$\Sigma = \begin{pmatrix} 1 & .5 & .15 & .15 & 0 & 0 \\ .5 & 1 & .15 & .15 & 0 & 0 \\ .15 & .15 & 1 & .25 & 0 & 0 \\ .15 & .15 & .25 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & .10 \\ 0 & 0 & 0 & 0 & .10 & 1 \end{pmatrix}$$

Norms

- Used for measuring the size of a vector
- Norms map vectors to non-negative values
- Norm of vector $\mathbf{x} = [x_1, \dots, x_n]^T$ is distance from origin to $\mathbf{x}$
 - It is any function f that satisfies:

$$f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = 0$$

$$f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y}) \quad \text{Triangle Inequality}$$

$$\forall \alpha \in \mathbb{R} \quad f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$$

NumPy Equivalent:

```
>>> x = np.array([[6., 9.]])
>>> y = np.array([[14., 18.]])
>>> np.linalg.norm(x + y)
33.60059523282288
>>> np.linalg.norm(x) + np.linalg.norm(y)
33.620162328374725
>>> █
```

L^p Norm

`linalg.norm(x, ord=None, axis=None, keepdims=False)`

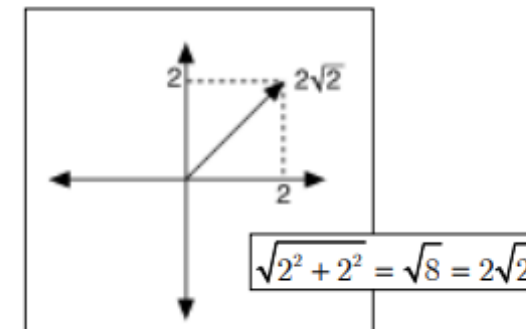
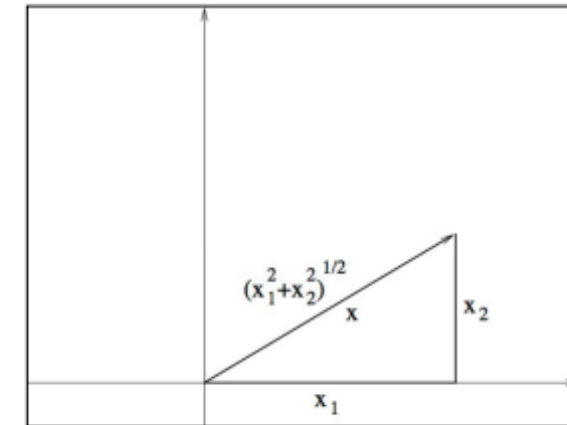
- Definition:

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

- L^2 Norm

- Called Euclidean norm

- Simply the Euclidean distance between the origin and the point x
 - written simply as $\|x\|$
 - Squared Euclidean norm is same as $x^T x$



- L^1 Norm

- Useful when 0 and non-zero have to be distinguished

- Note that L^2 increases slowly near origin, e.g., $0.1^2=0.01$)

- L^∞ Norm $\|x\|_\infty = \max_i |x_i|$

- Called max norm

The Frobenius norm

Similar to L^2 norm

$$\|A\|_F = \left(\sum_{i,j} A_{ij}^2 \right)^{\frac{1}{2}}$$

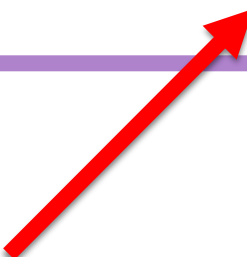
$$A = \begin{bmatrix} 2 & -1 & 5 \\ 0 & 2 & 1 \\ 3 & 1 & 1 \end{bmatrix} \quad \|A\| = \sqrt{4 + 1 + 25 + \dots + 1} = \sqrt{46}$$

Norms Can Serve as the Building Blocks for Distance Measurements!

- Distance between two vectors (v, w)

$$\text{-- dist}(v, w) = \|v - w\|$$

$$= \sqrt{(v_1 - w_1)^2 + \dots + (v_n - w_n)^2}$$



Euclidean distance

Inversion: use numpy.linalg

A

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$A^{-1}A = I_n$$

Standard inversion operator:

```
>>> np.linalg.inv(X)
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
>>> 
```

Determinant, i.e., $\det(A) = |A|$

$$Ax = b$$

$$A^{-1}Ax = A^{-1}b$$

$$I_n x = A^{-1}b$$

$$x = A^{-1}b$$

(Moore-Penrose) pseudo-inverse operator:

```
>>> np.linalg.pinv(X)
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
```

Sometimes, there is no matrix inverse!

Linear Algebra Toolset

- You know tensors now, i.e., how to collect numbers into constructs
- You have a bunch of operators to do on these tensors
 - Elementwise math
 - Re-shaping, transposing
 - Aggregation functions (non-elementwise simple operators)
 - Matrix multiplication, inversion
 - Norms and distance functions
- We'll develop more tools as needed
 - Determinants
 - Singular value decomposition
 - Approximate inversion (pseudo-inversion)
 - More special types of tensors/matrices

QUESTIONS?

Deep robots!

Deep questions?!

