Machine Learning:
Elements of **Linear Algebra**

Alexander G. Ororbia II

Introduction to Machine Learning

CSCI-635

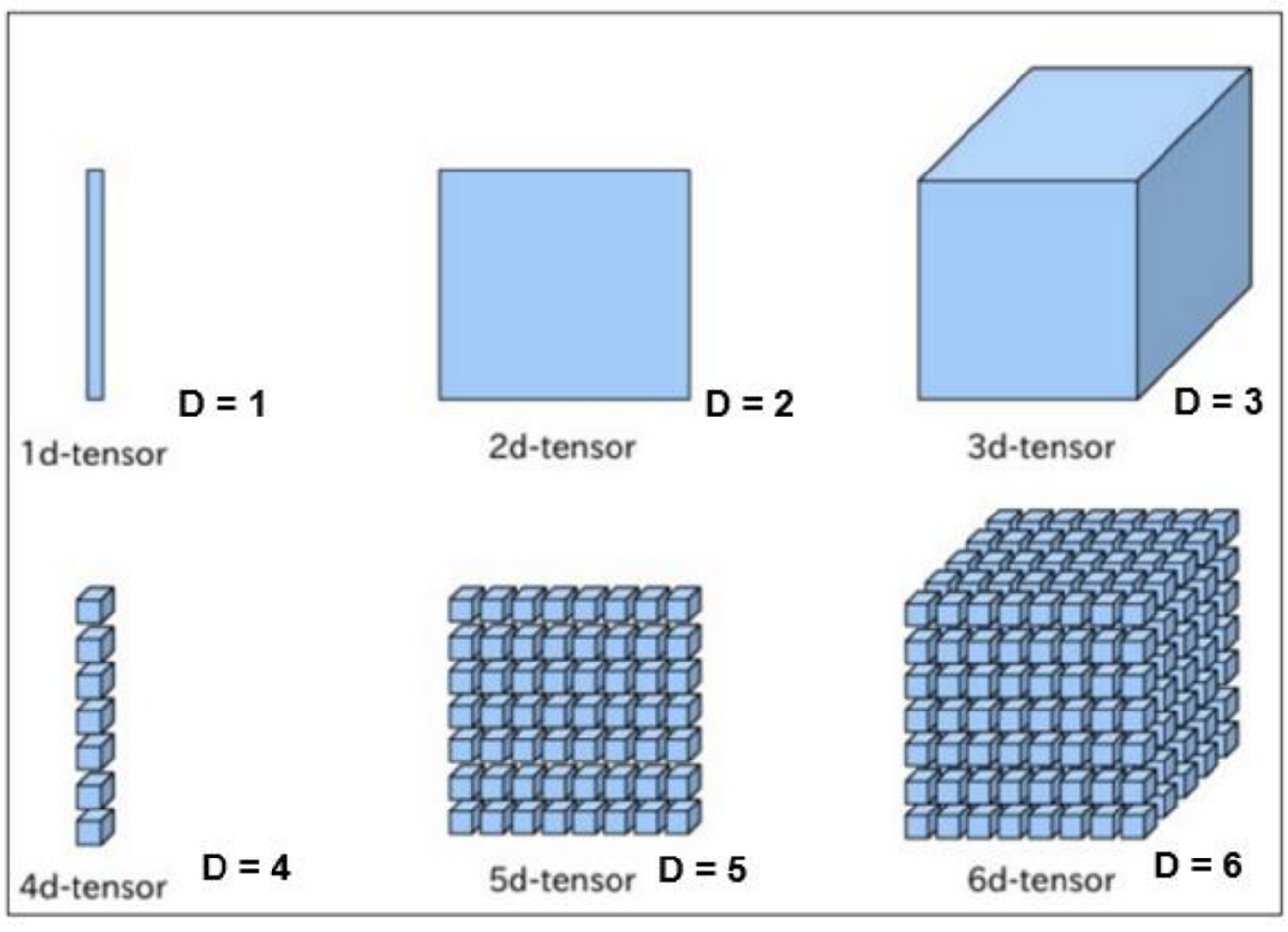
1/14/2026 and 1/16/2026

Special thanks to: Sargur N. Srihari

So, Again, What are Tensors?

```
import numpy as np
```

- A building block of linear algebra
- A mathematical formalism for a multi-dimensional collection / object – houses items, notably numbers / values
  - In Python, there are often called arrays or n-dimensional arrays (*ndarrays*)



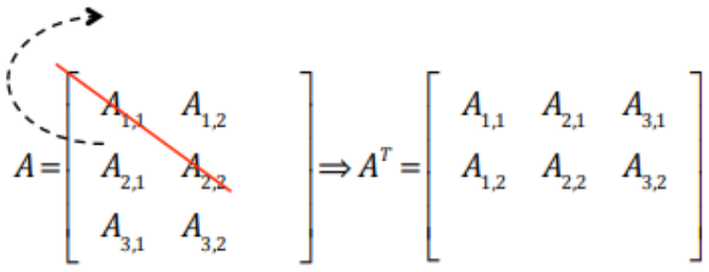
Transposing/manipulation in NumPy

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

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

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.



Hadamard Product

- Multiply each A(i, j) to each corresponding B(i, j)
  - Element-wise multiplication

0.5	-0.7
-0.69	1.8

$\odot$

0.5	-0.7
-0.69	1.8

=

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

Elementwise Functions

- Applied to each element (i, j) of matrix/vector argument
  - Could be *cos(.)*, *sin(.)*, *tanh(.)*, etc. (the "." means argument)
- *Identity*: $\phi(\mathbf{v}) = \mathbf{v}$
- *Logistic Sigmoid*: $\phi(\mathbf{v}) = \sigma(\mathbf{v}) = \frac{1}{1+e^{-\mathbf{v}}}$
- *Softmax*: $\phi(\mathbf{v}) = \frac{\exp(\mathbf{v})}{\sum_{c=1}^{C} \exp(\mathbf{v}_c)}$ $\mathbf{v} \in \mathbb{R}^C$
- *Linear Rectifier*: $\phi(\mathbf{v}) = \max(0, \mathbf{v})$

$$\varphi\left(\begin{array}{|c|c|}\hline 1.0 & -1.4 \\ \hline -0.69 & 1.8 \\ \hline\end{array}\right) =$$

φ(1.0) = 1	φ(-1.4) = 0
φ(-0.69) = 0	φ(1.8) = 1.8

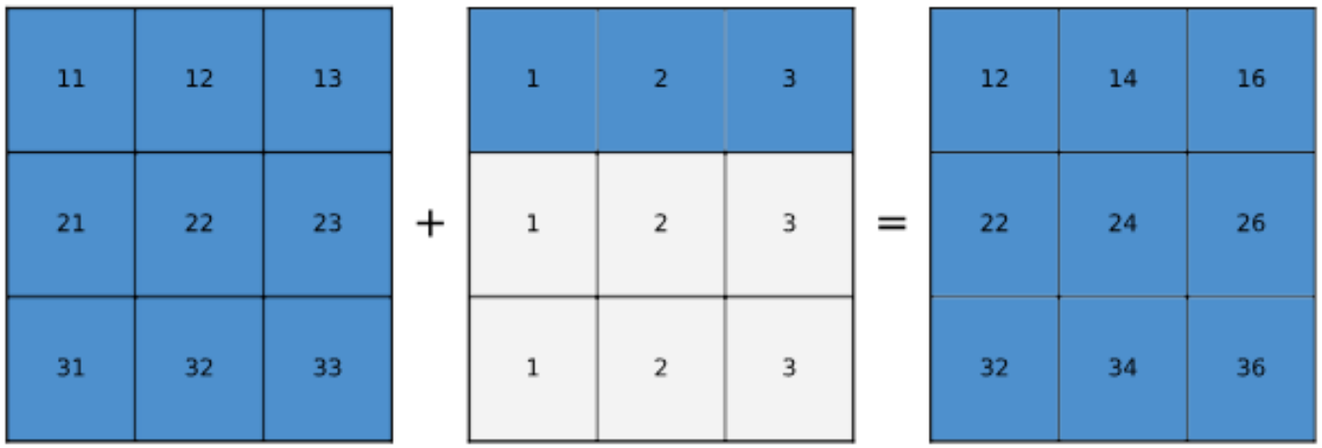
Elementwise Operators

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

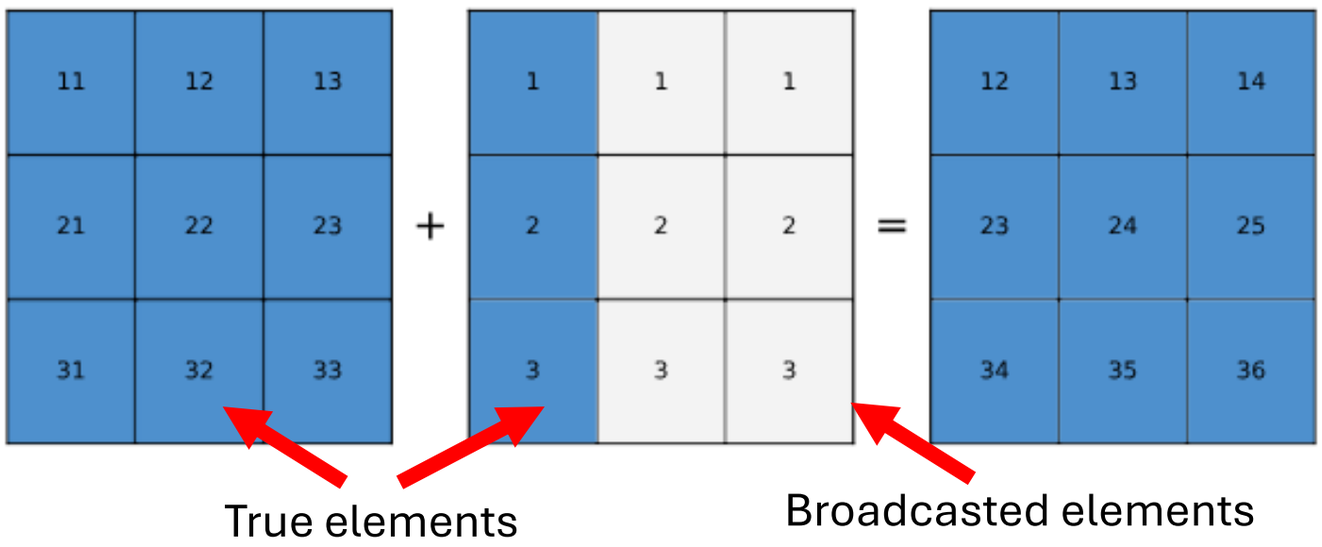
Operator	Operation
+, +=	Addition
-, -=	Subtraction
*, *=	Multiplication
/, /=	Division
//, //=	Integer division
**, **=	Exponentiation

Vectorization and Broadcasting

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



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



- ***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 ae 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

Elementwise / Broadcasted 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) “*” ($\odot$)

0.5	-0.7
-0.69	1.8

$\odot$

0.5	0.7
-0.69	1.8

=

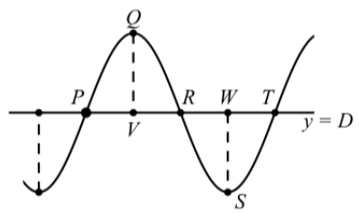
.5 * .5 = .25	-.7 * .7 = -.49
-.69 * -.69 = .4761	1.8 * 1.8 = 3.24

- **Division** “/” (/)

Elementwise Functions (Vectorized Operators)

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

NumPy Function	Description
`np.cos, np.sin, np.tan`	Trigonometric functions
`np.arccos, np.arcsin, np.arctan`	Inverse trigonometric functions
`np.cosh, np.sinh, np.tanh`	Hyperbolic trigonometric functions
`np.arccosh, np.arcsinh, np.arctanh`	Inverse hyperbolic trigonometric functions
`np.sqrt`	Square root
`np.exp`	Exponential
`np.log, np.log2, np.log10`	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
`np.add, np.subtract, np.multiply, np.divide`	Addition, subtraction, multiplication, and division of two NumPy arrays
`np.power`	Raises first input argument to the power of the second input argument (applied elementwise)
`np.remainder`	The remainder of the division
`np.reciprocal`	The reciprocal (inverse) of each element
`np.real, np.imag, np.conj`	The real part, imaginary part, and the complex conjugate of the elements in the input arrays
`np.sign, np.abs`	The sign and the absolute value
`np.floor, np.ceil, np.rint`	Converts to integer values
`np.round`	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
`np.sum`	The sum of all elements
`np.prod`	The product of all elements
`np.min, np.max`	The minimum/maximum value in an array
`np.argmin, np.argmax`	The index of the minimum/maximum value in an array

Σ = summation (capital sigma)

Π = product (capital pi)

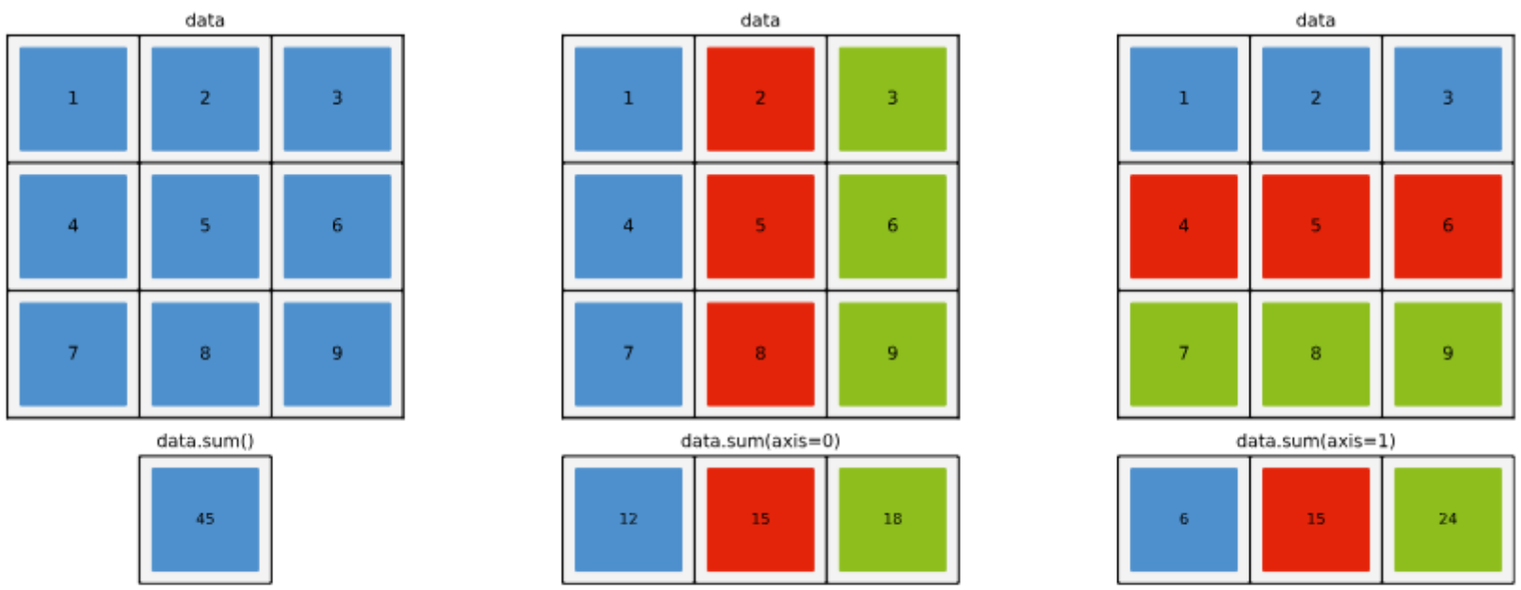


Figure 2-3. *Illustration of array aggregation functions along all axes (left), the first axis (center), and the second axis (right) of a two-dimensional array of shape 3 × 3*

Elementwise Composed Functions

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

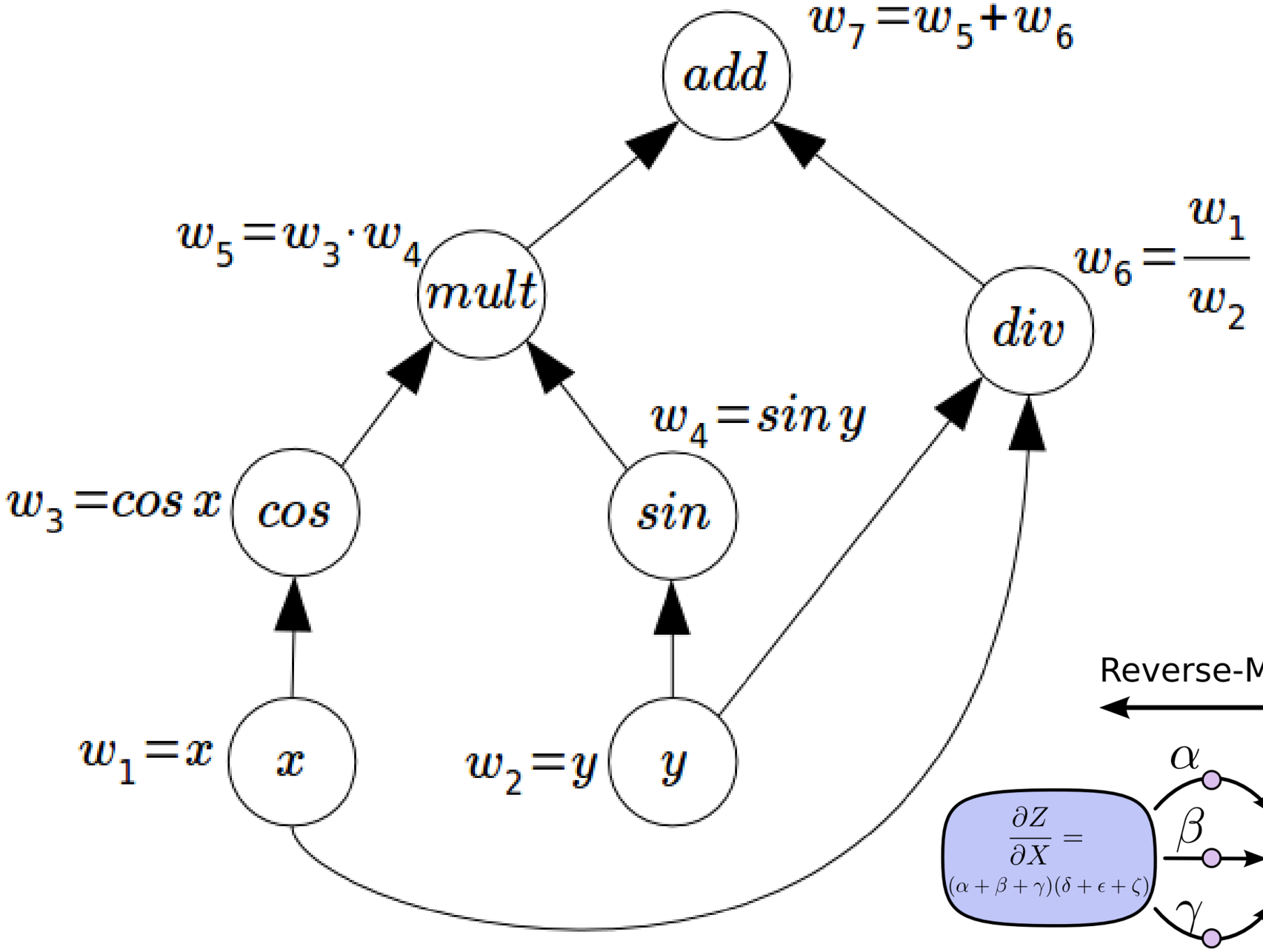
***Softmax*:** $\phi(\mathbf{v}) = \frac{\exp(\mathbf{v})}{\sum_{c=1}^{C} \exp(\mathbf{v}_c)}$ ***Sigmoid*:** $\phi(\mathbf{v}) = \sigma(\mathbf{v}) = \frac{1}{1+e^{-\mathbf{v}}}$

Let's write these out to our Python interpreter!

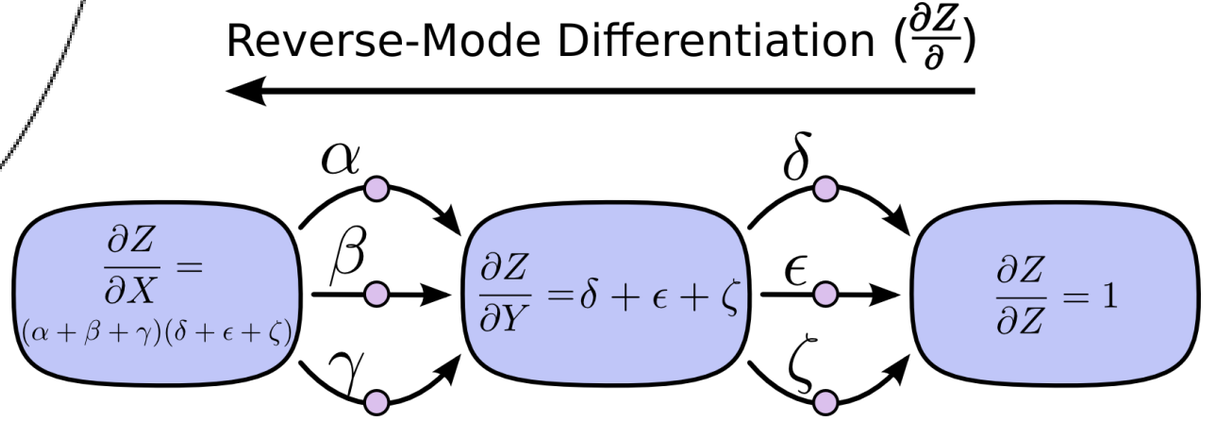
NumPy Function
np.cos, np.sin, np.tan
np.arccos, np.arcsin, np.arctan
np.cosh, np.sinh, np.tanh
np.arccosh, np.arcsinh, np.arctanh
np.sqrt
np.exp
np.log, np.log2, np.log10

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

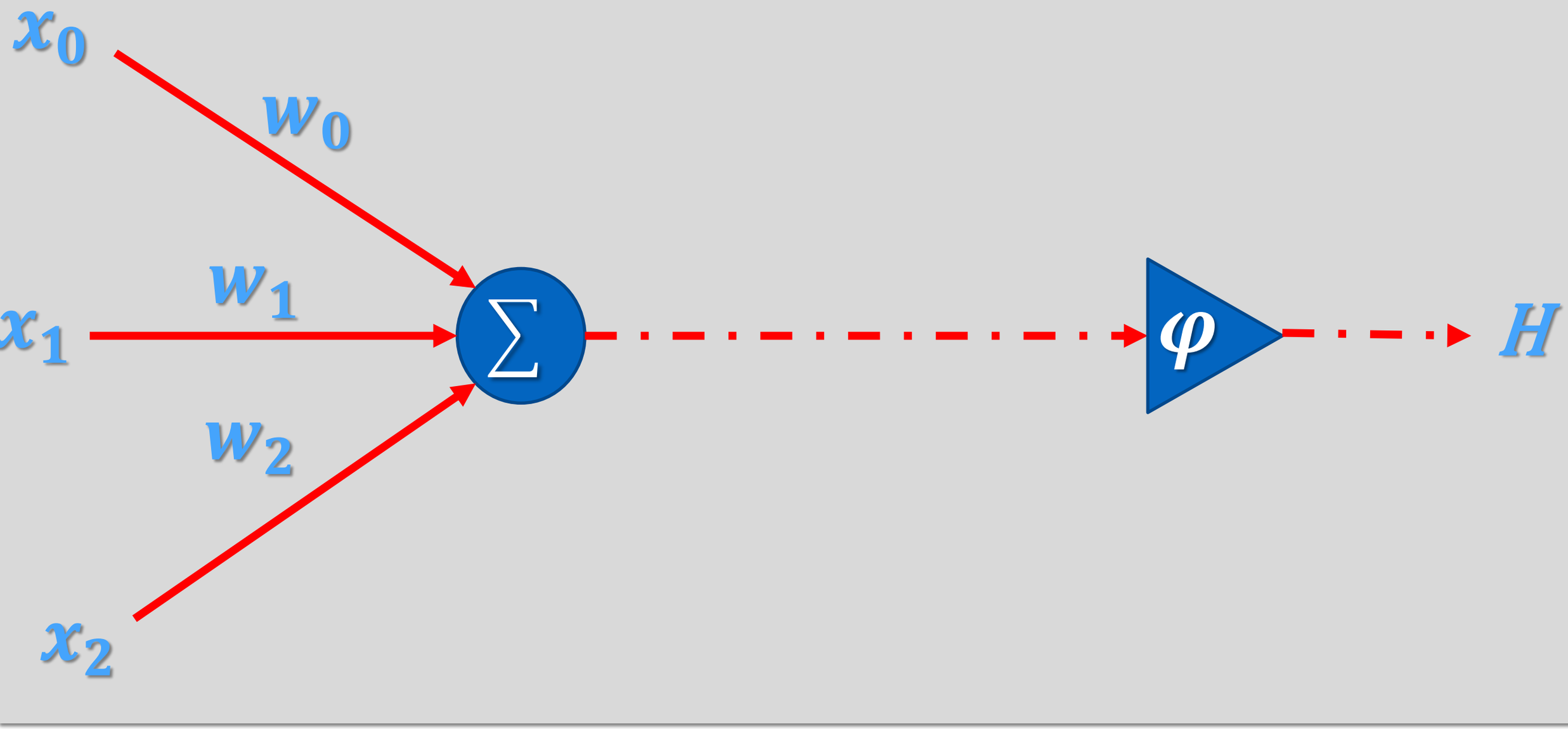
Why Do We Care?
Computation Graphs



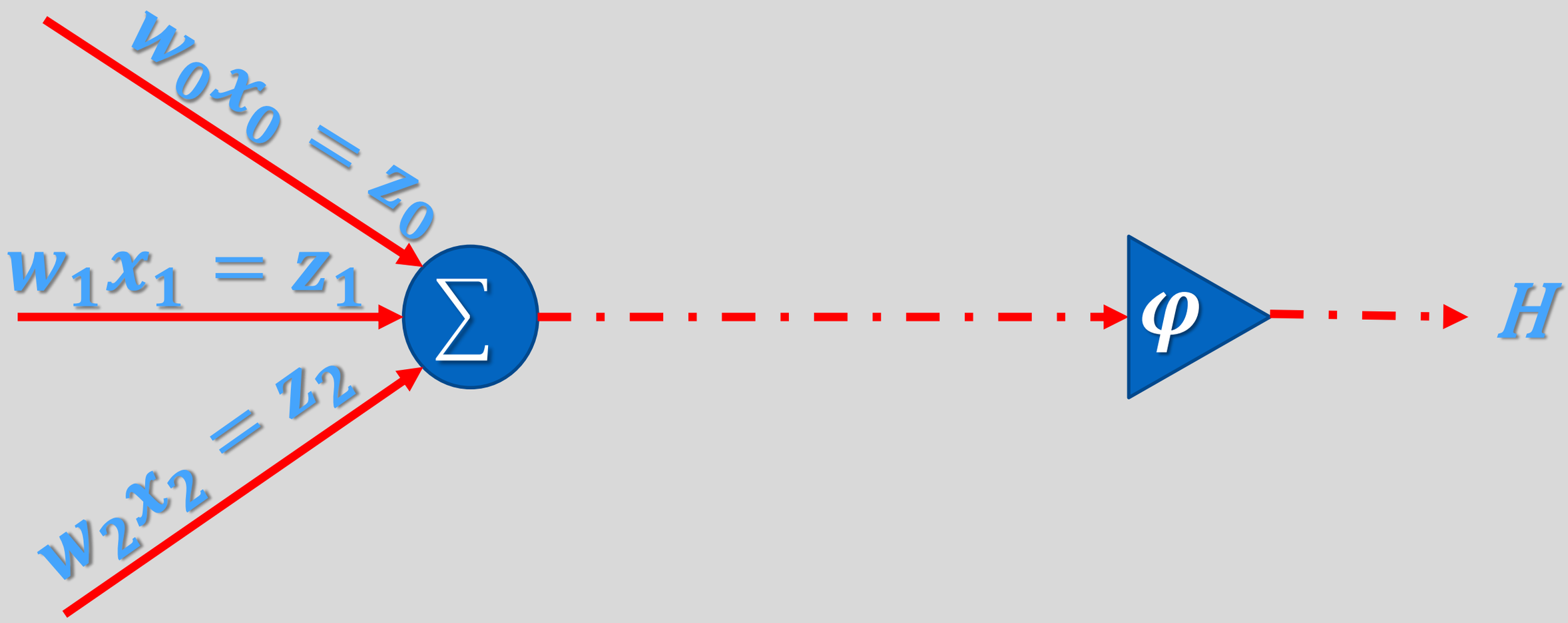
Linear algebra operators can be arranged in a directed graph!



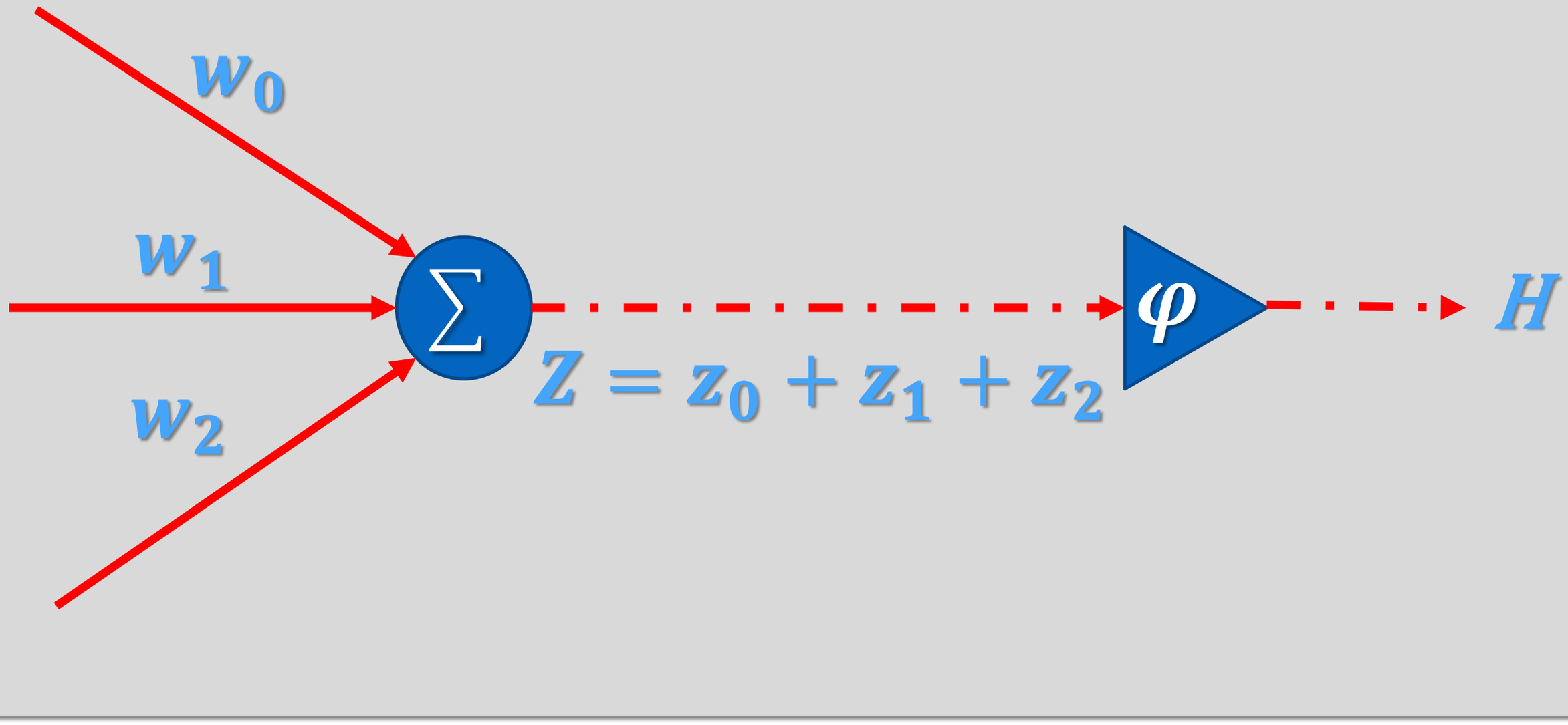
A Simple Linear Predictor



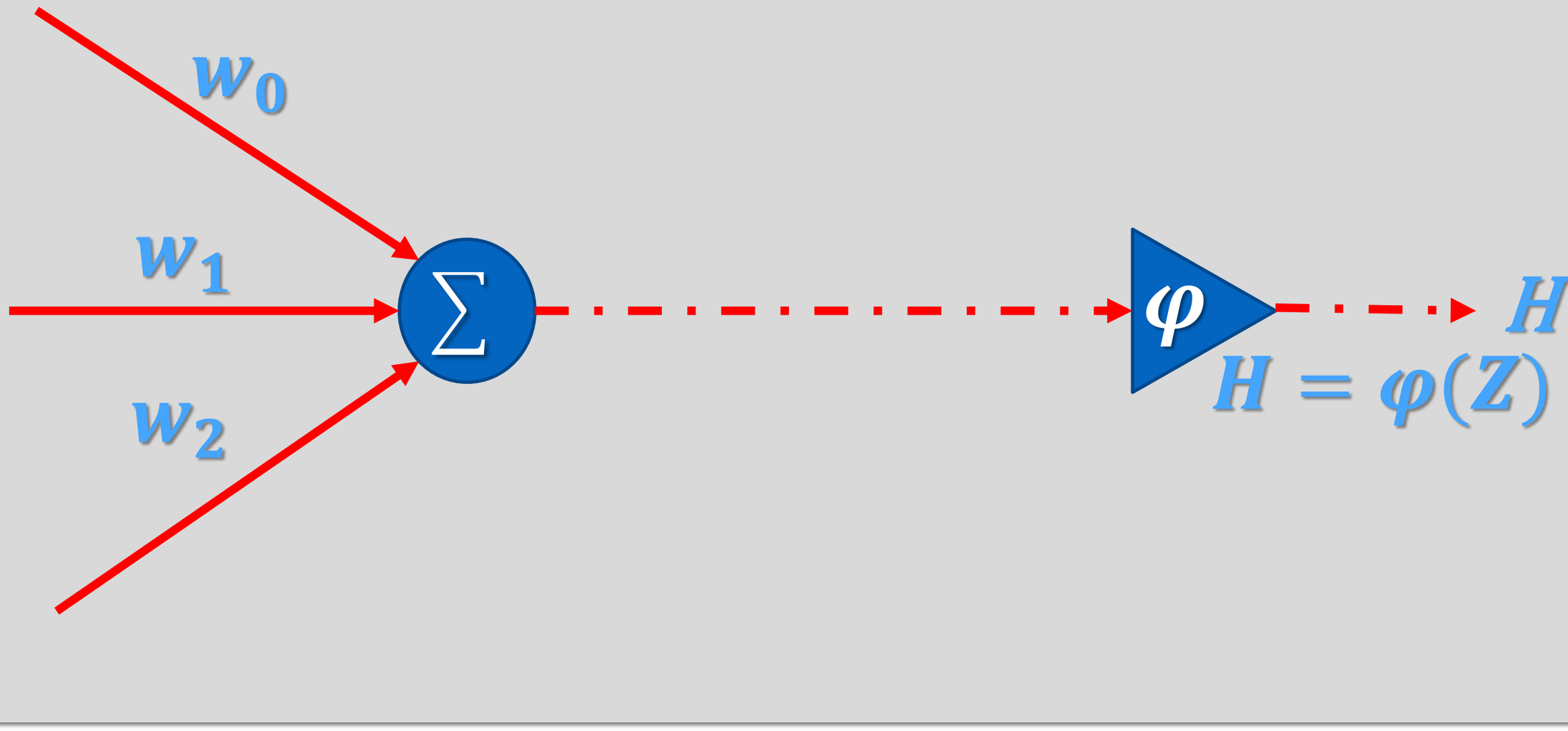
A Simple Linear Predictor



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:

w_0	w_1	w_2

*

x_0
x_1
x_2

$= \varphi(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)$

Vector Form (Two Units)

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

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

Known as vectorization!

Now Let Us Fully Vectorize This!

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

h_0:	w_0	w_1	w_2	*	x_0	x_3	=	$\varphi(w_0 * x_0 + \ldots)$	$\varphi(w_0 * x_3 + \ldots)$
h_1:	w_3	w_4	w_5		x_1	x_4		$\varphi(w_3 * x_0 + \ldots)$	$\varphi(w_3 * x_3 + \ldots)$
					x_2	x_5			

Vectorized Operations / Objects

```
import numpy as np
```

- Vectors, matrices, tensors (greater than 2nd order arrays)
  - Essential tool for numerical computation
  - Useful data representation, especially for operations (ops) that are repeated for set of values
- Vectorization – eliminates the need for explicit loops (batch operations applied to data)
  - In Python, these ops often built on lower-level libraries (XLA, LAPACK, etc.)
- Essentially what NumPy, i.e., $numpy$ (and $numpy.linalg$), is for
  - Built on C (for processing/manipulating arrays)
  - All elements in a numpy (fixed-size) array are of the same type (homogeneous)
  - These arrays have in-built operations to be applied to them (along with functions/modules that work with these data structures)

Identity Matrices

- Matrix inversion is a powerful tool to analytically solve $\boldsymbol{Ax}=\boldsymbol{b}$
- 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 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

18

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.]])
```

Norms

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

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

$$f(\boldsymbol{x}+\boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y}) \qquad \text{Triangle Inequality}$$

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

NumPy Equivalent:

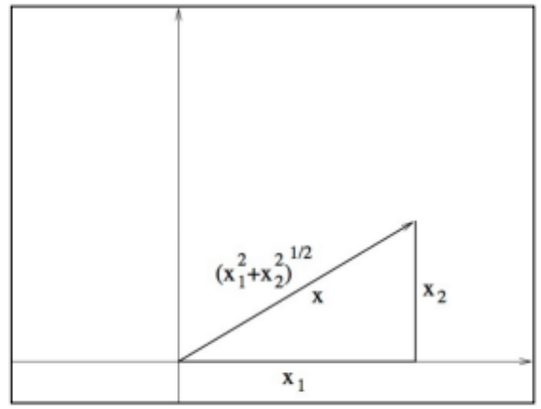
```
>>> x = np.array([[6., 9.]])
>>> y = np.array([[14., 18.]])
>>> np.linalg.norm(x + y)
33.60059523282288
>>> np.linalg.norm(x) + np.linalg.norm(y)
33.620162328374725
>>>
```

L^p Norm

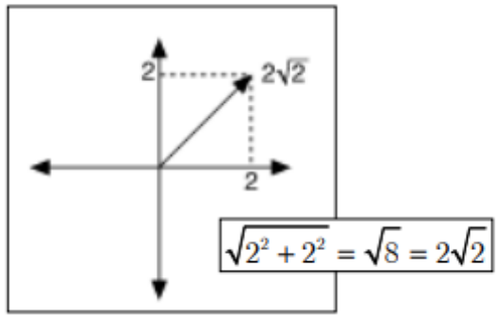
`linalg.norm(x, ord=None, axis=None, keepdims=False)`

- Definition:

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



  - L^2 Norm
    - Called Euclidean norm
      - Simply the Euclidean distance between the origin and the point $\boldsymbol{x}$
      - written simply as $||\boldsymbol{x}||$
      - Squared Euclidean norm is same as $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$



  - 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 $\left\|\boldsymbol{x}\right\|_\infty = \max_i \left|x_i\right|$
    - Called max norm

The Frobenius norm

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

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

Norms Can Serve as the Building Blocks for Distance Measurements!

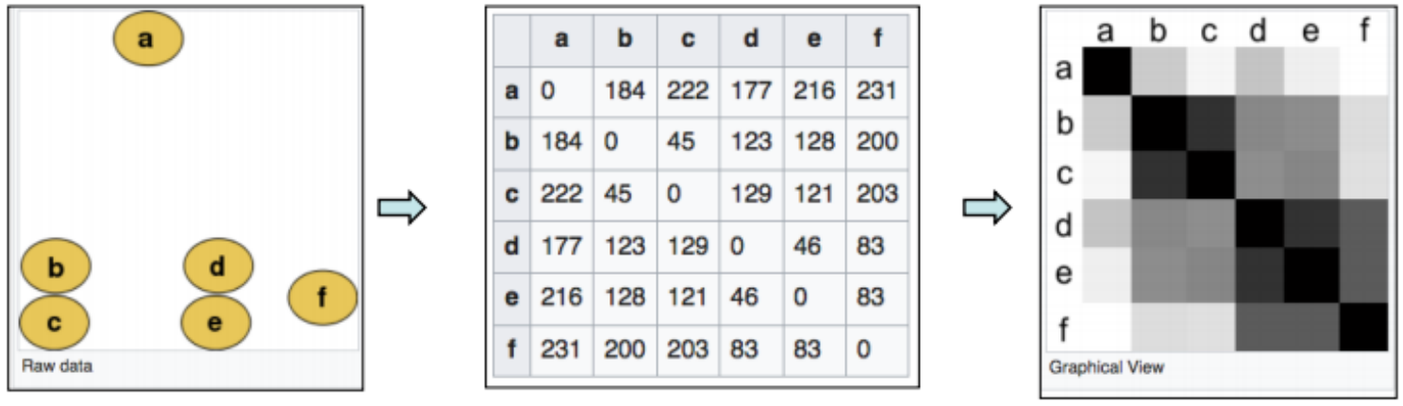
- Distance between two vectors $(\boldsymbol{v},\boldsymbol{w})$
  - $\mathrm{dist}(\boldsymbol{v},\boldsymbol{w}) = ||\boldsymbol{v}\text{-}\boldsymbol{w}||$

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

Euclidean distance

The Symmetric Matrix

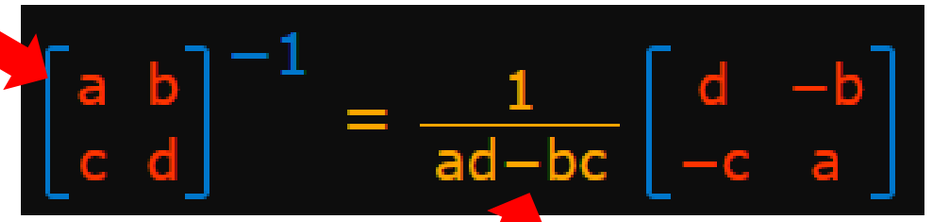
- A symmetric matrix equals its transpose: $A = A^T$
  - E.g., a distance matrix is symmetric with $A_{ij} = A_{ji}$



  - E.g., covariance matrices are symmetric

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

Inversion: use numpy.linalg



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

Standard inversion operator:

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

$$Ax = b$$

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

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

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

(Moore-Penrose) pseudo-inverse operator:

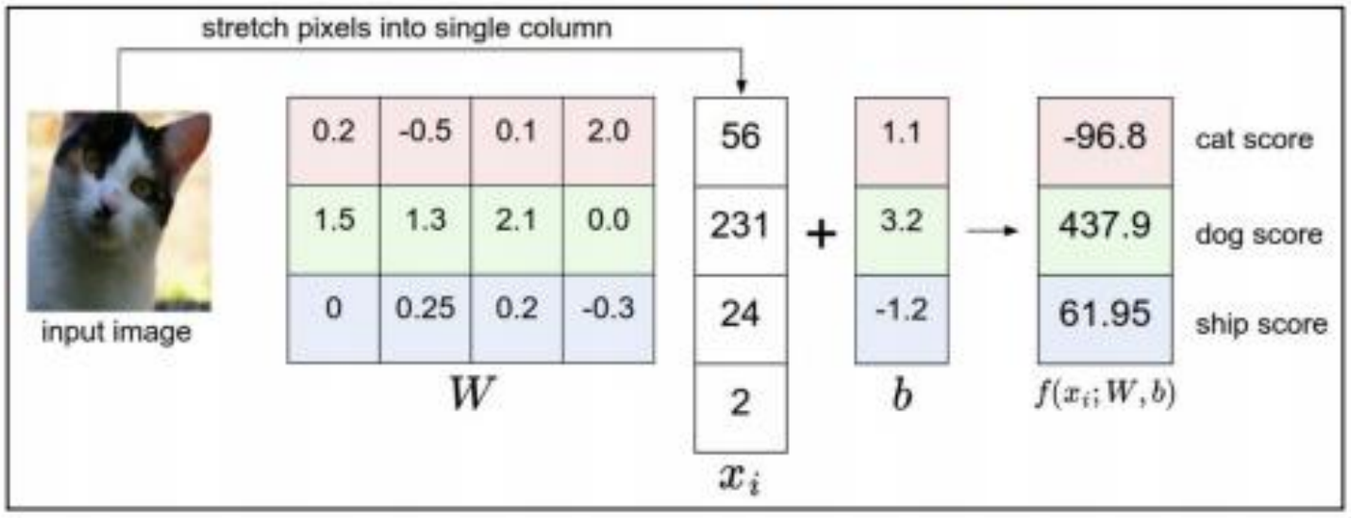
```
>>> np.linalg.pinv(X)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Sometimes, there is no matrix inverse!

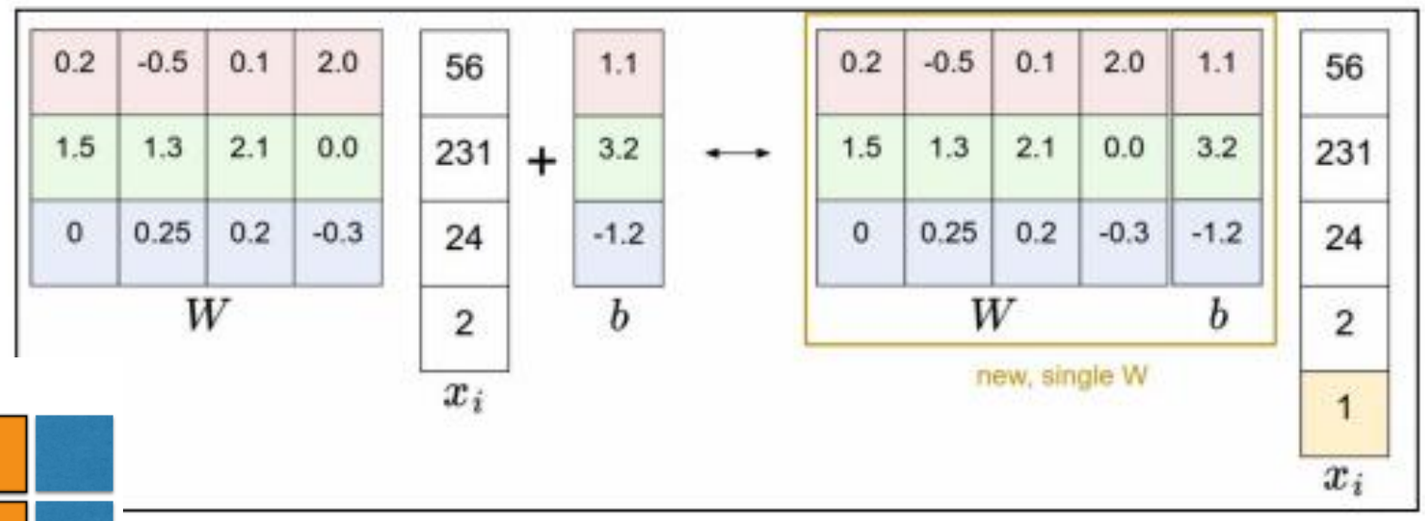
Tensors in Machine Learning

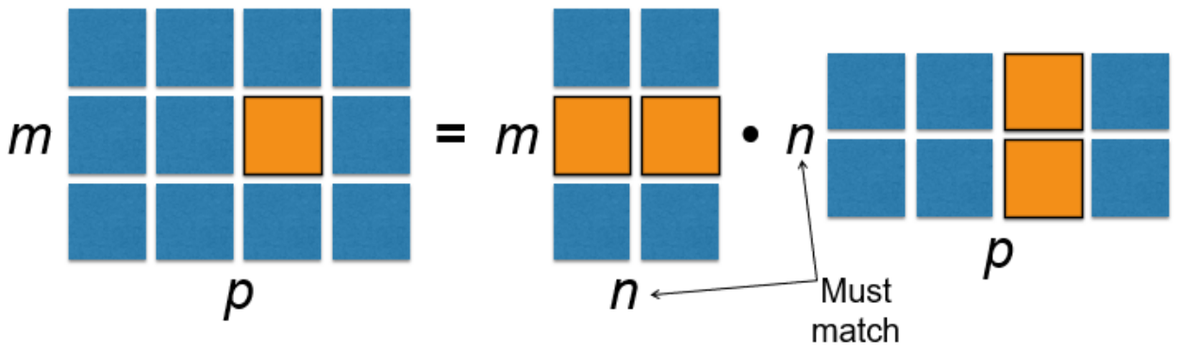
Vector x is converted into vector y by multiplying x by a matrix W

A linear classifier $y = Wx + b$

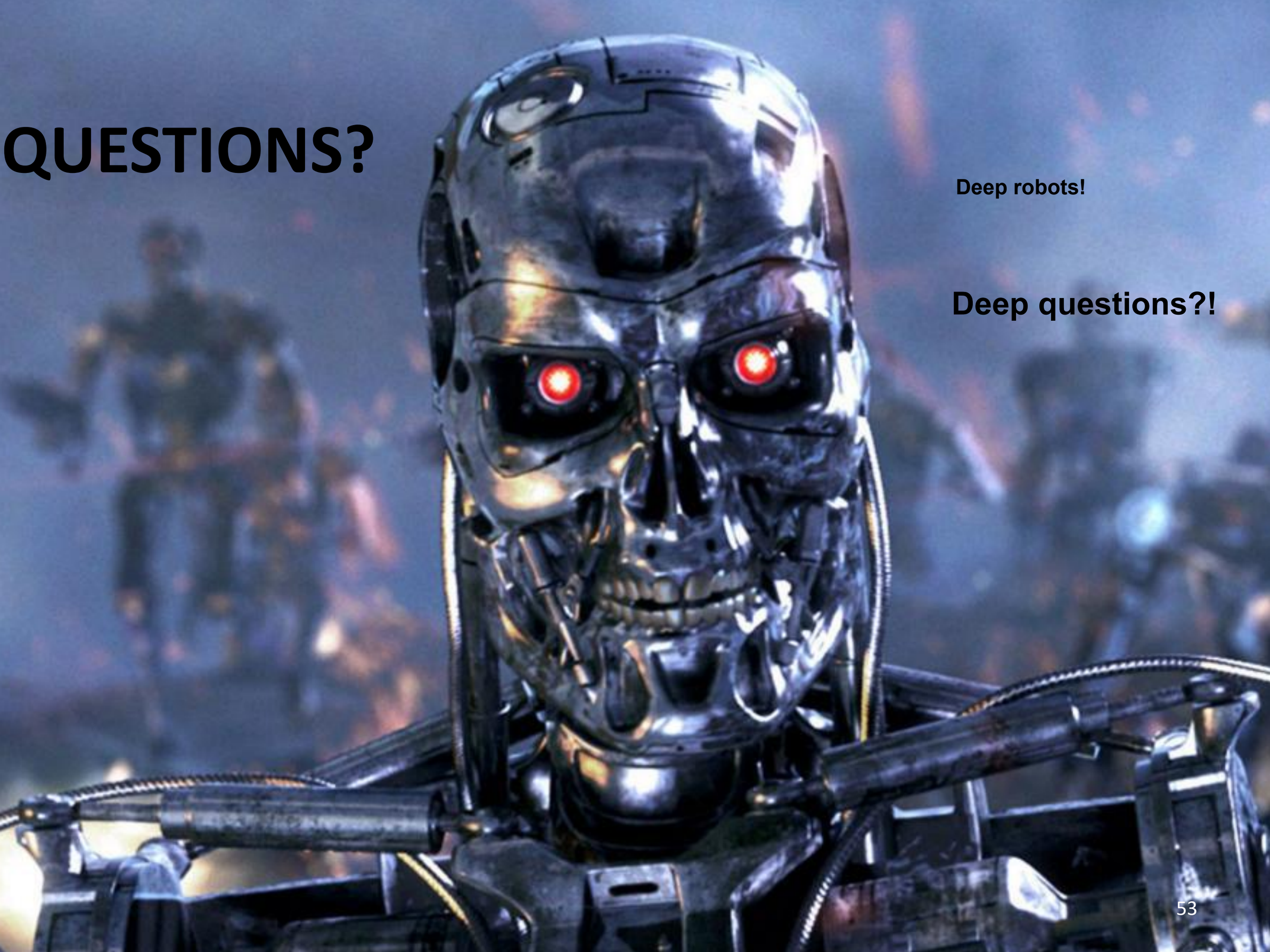


A linear classifier with bias eliminated $y = Wx$





QUESTIONS?



Deep robots!

Deep questions?!