Machine Learning:
Elements of **Linear Algebra**

Alexander G. Ororbia II, William Gebhardt

Introduction to Machine Learning

CSCI-635

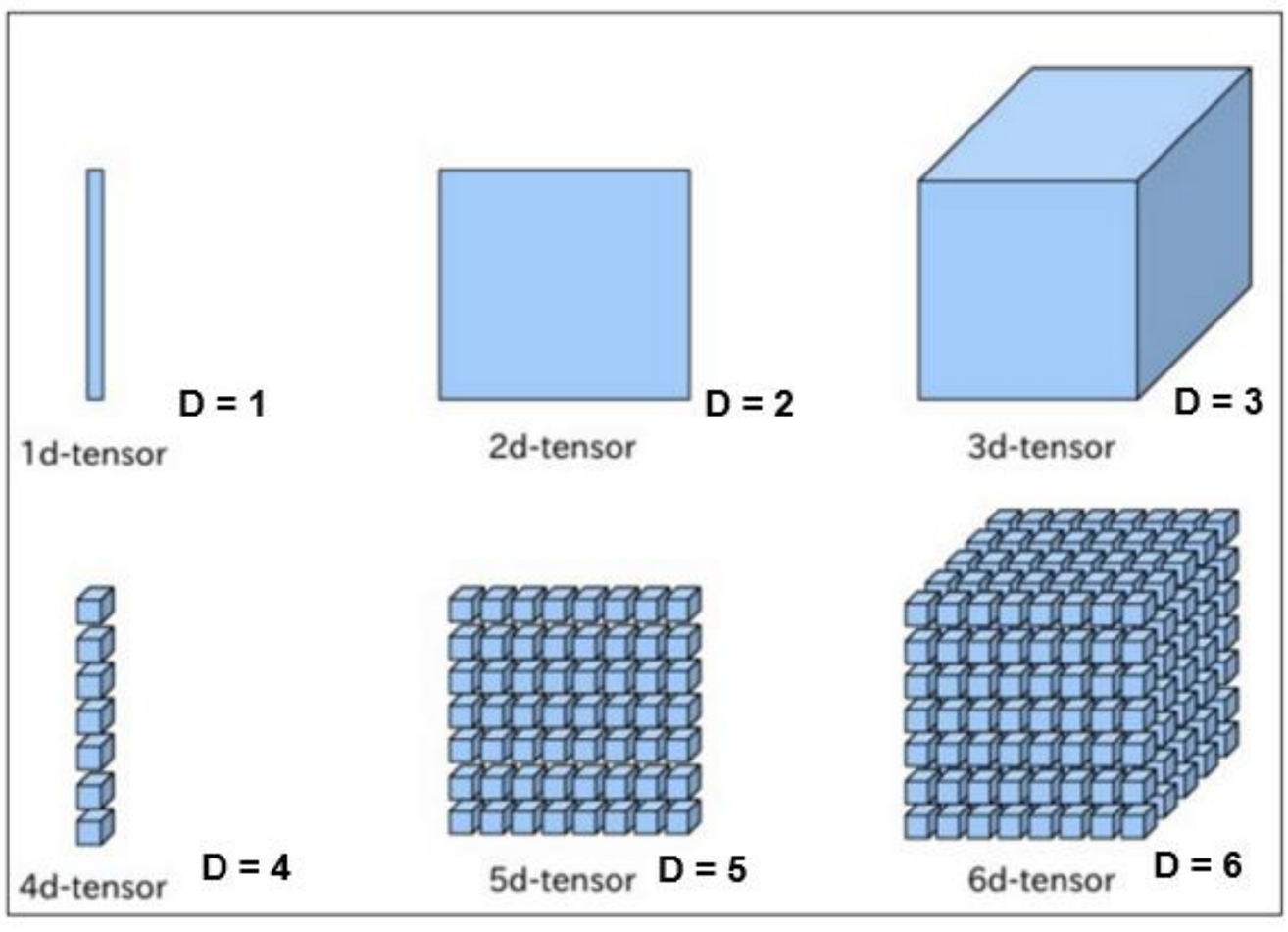
1/14/2026 and 1/16/2026

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*)



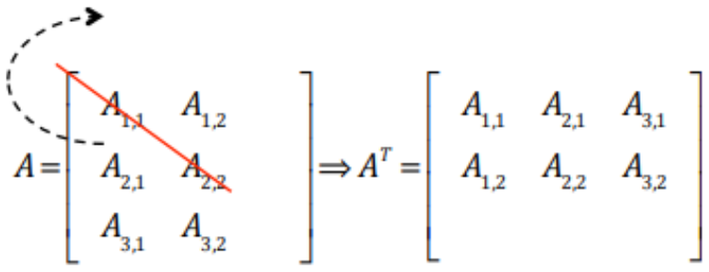
Transposing/manipulation in NumPy

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

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

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

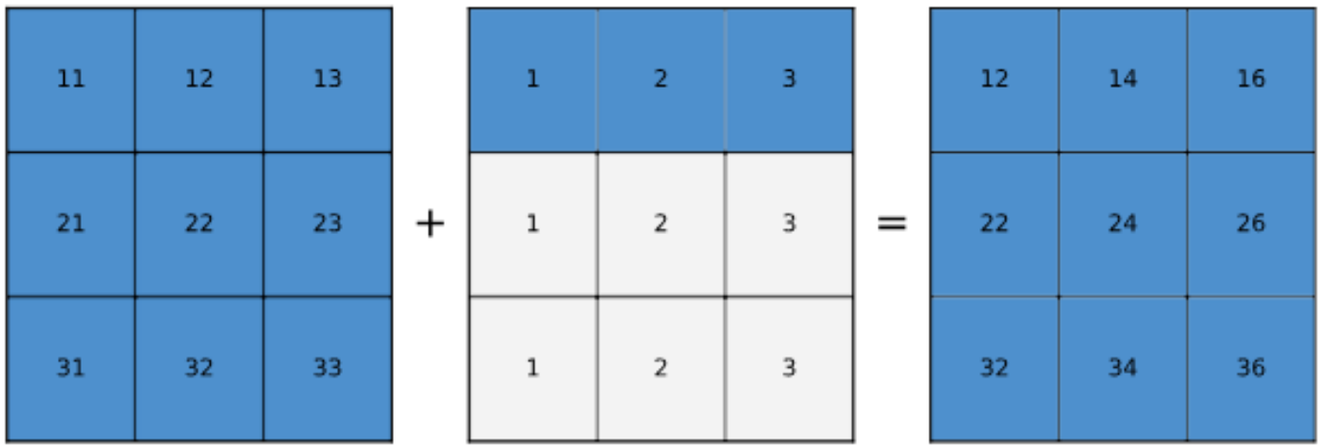
Elementwise Operators

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

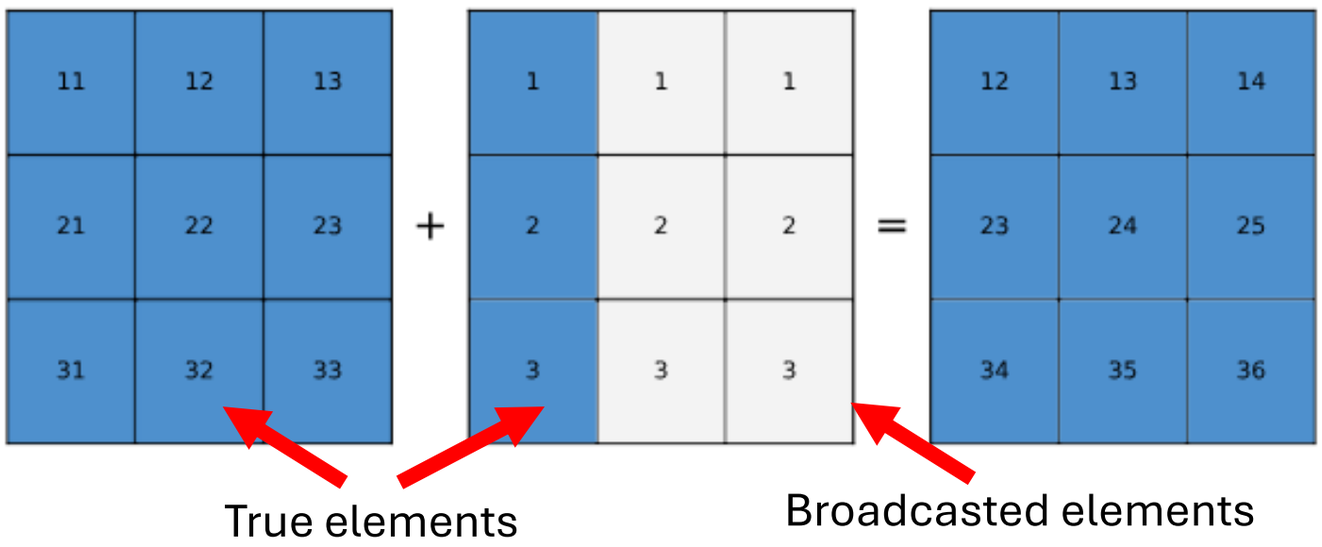
Operator	Operation
+, +=	Addition
-, -=	Subtraction
*, *=	Multiplication
/, /=	Division
//, //=	Integer division
**, **=	Exponentiation

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

=

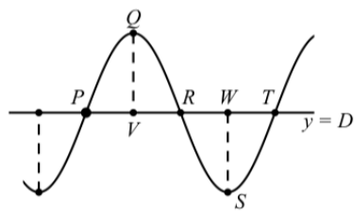
.5 * .5 = .25	-.7 * .7 = -.49
-.69 * -.69 = .4761	1.8 * 1.8 = 3.24

- **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

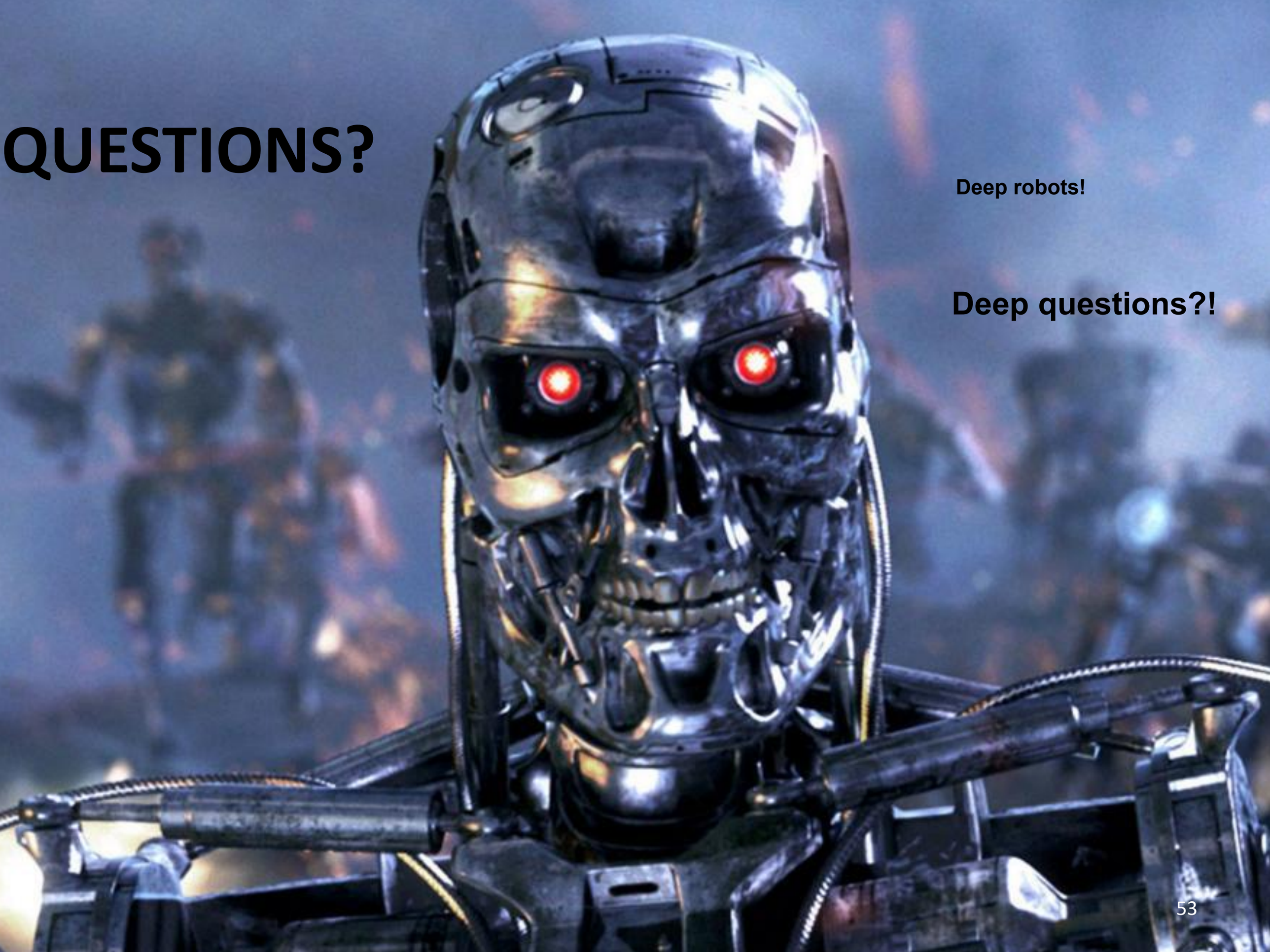
- NumPy contains many vectorized functions – elementwise evaluation of basic mathematical functions (over tensors)
  - Take in tensor/ndarray, return tensor/ndarray of same shape as input (though not necessarily of same type)

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

QUESTIONS?



Deep robots!

Deep questions?!