Learning Deep Learning
Chapter 1b

Chao-En Yu

National Tsing Hua University

Linear Algebra

- Knowledge of linear algebra is handy when working with neural networks.
- Computing dot products, matrix-vector multiplications, and matrix multiplications efficiently is important in many scientific fields.
- So much effort has been spent on creating efficient implementations of these operations, implemented by a Python package known as NumPy.

NumPy

functions or procedures →

- NumPy uses Basic Linear Algebra Subprograms (BLAS) which are typically implemented in Fortran or C and heavily optimized to run as fast as possible.

- Further, if you have a graphics processing unit (GPU) capable of running CUDA, there are the CUDA libraries that enable NumPy to perform these operations efficiently on the GPU.

Vector

- Specifying mathematical problems in vector or matrix form enables us to take advantage of efficient mathematics library implementations, and in particular to offload computations to the GPU/TPU.

- We can arrange input/weight values into a single input/weight *vector* variable (***bold italic***):

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}, \qquad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix}.$$

We start subscripts at 0 to stay consistent when translating formulas into Python code.

Transpose and Addition

- The above vectors are known as column vectors. We use the transpose operation to convert it into a row vector, denoted by $\boldsymbol{x}^T$.

$$\boldsymbol{x}^T = (x_0 \quad x_1 \quad \dots \quad x_n)$$

- Vector addition is an *element-wise operation*:

$$\boldsymbol{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix} \Rightarrow \boldsymbol{a} + \boldsymbol{b} = \begin{pmatrix} a_0 + b_0 \\ a_1 + b_1 \\ \vdots \\ a_n + b_n \end{pmatrix}.$$

Dot Product

- The dot product is defined only if the two vectors are of equal length, just like vector addition.

- It is computed by multiplying element 0 of the two vectors, then multiplying element 1 of the two vectors and so on, and finally adding all of these products:

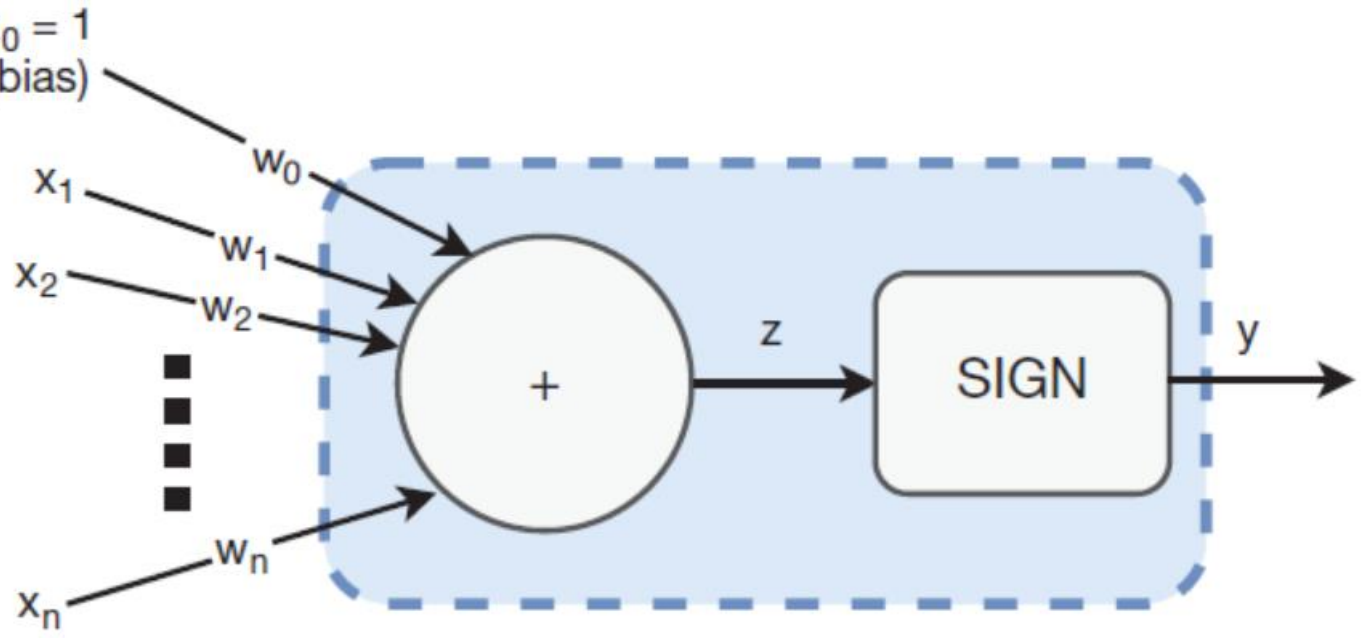
$$\boldsymbol{w} \cdot \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n = \sum_{i=0}^{n} w_i x_i$$

Perceptron Using Dot Product

- We can write our perceptron function using the NumPy dot-product functionality:

```
import numpy as np
w=[0.9, -0.6, -0.5]; x=[1,-1,-1];
z=np.dot(x, w)
y_hat = np.sign(z)
```

There are $n + 1$ inputs.



Matrix

- A two-dimensional (2D) structure has two axes which is known as a *matrix*. An example of a matrix $\boldsymbol{W}$ with $m + 1$ rows and $n + 1$ columns is shown here:

$$\boldsymbol{W} = \begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & \cdots & w_{mn} \end{pmatrix}$$

- The elements in the vertical direction are numbered in increasing order going downward in a matrix (an increasing y-value in an xy chart is increasing going upward).

Matrix Transpose

- Recall that the weights ($\boldsymbol{w}$) for a single neuron can be represented by a vector.
- The weights for $m + 1$ neurons (perceptrons) each with $n + 1$ input features/weights can be arranged in an array of vectors, i.e., a matrix.
- We transpose a matrix by flipping the matrix along its diagonal: element ij becomes element ji.

How to create $\boldsymbol{W}$ in Python? →

$$\boldsymbol{W} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \qquad \boldsymbol{W}^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

Matrix-Vector Multiplication

- We are ready to define matrix-vector multiplication:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} = \begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

The number of columns in the matrix need to match the number of elements in the vector.

$$= \begin{pmatrix} w_{00}x_0 + w_{01}x_1 + \cdots + w_{0n}x_n \\ w_{10}x_0 + w_{11}x_1 + \cdots + w_{1n}x_n \\ \vdots \\ w_{m0}x_0 + w_{m1}x_1 + \cdots + w_{mn}x_n \end{pmatrix}.$$

Two Unsupervised Classes

- Each element in the above resulting vector is:

$$z_i = \sum_{j=0}^{n} w_{ij} x_j = \boldsymbol{w}_i^T \cdot \boldsymbol{x}.$$

- The matrix-vector multiplication is doing $m + 1$ dot products of the matrix rows and the x-vector:

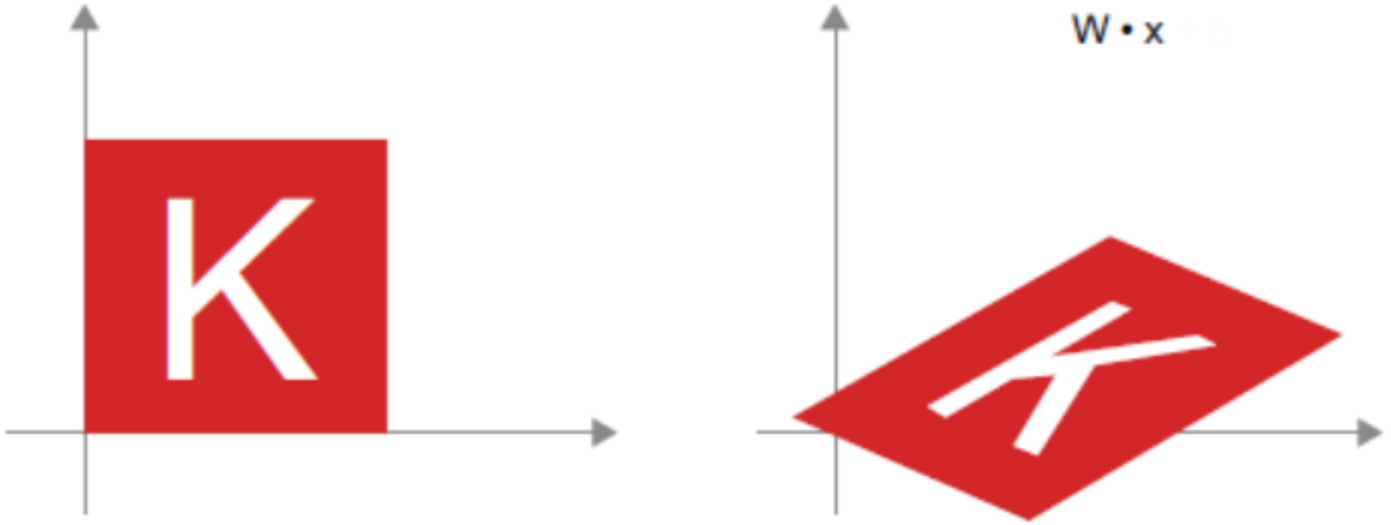
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{w}_0^T \\ \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_m^T \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_0^T \cdot \boldsymbol{x} \\ \boldsymbol{w}_1^T \cdot \boldsymbol{x} \\ \vdots \\ \boldsymbol{w}_m^T \cdot \boldsymbol{x} \end{pmatrix}.$$

Transformation

- The vector $\boldsymbol{z}$ now contains $m + 1$ elements. Each element represents the weighted sum for a single neuron presented with the single input example.

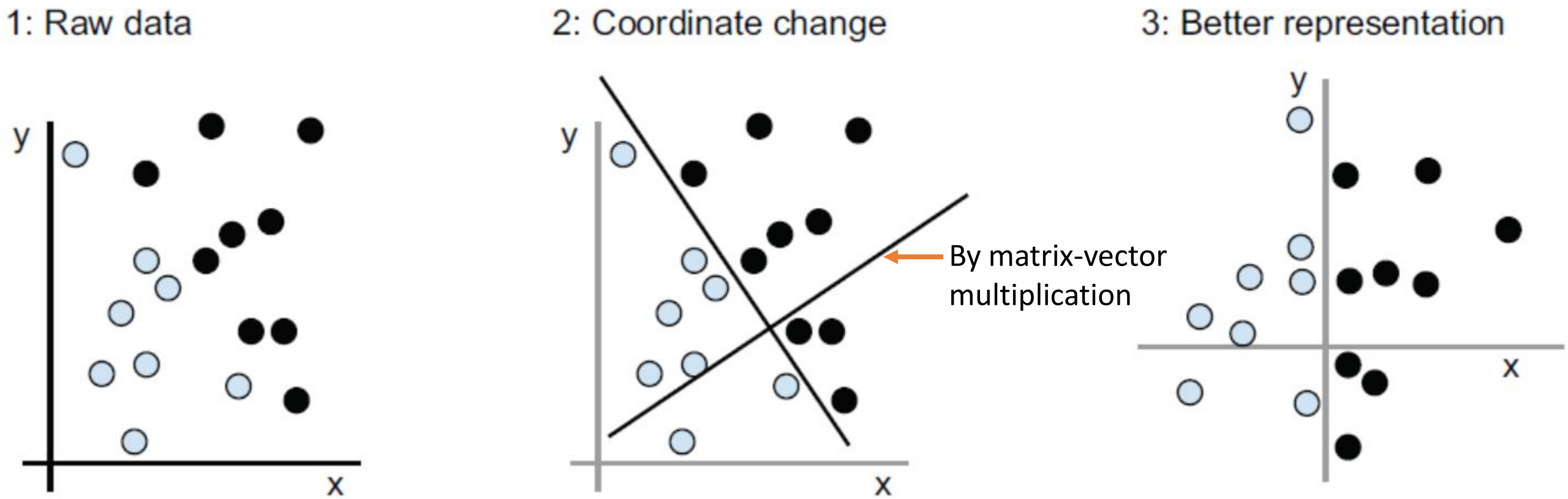
- Ex 1. $\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} =$

- Ex 2. $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} =$



Appropriate Representation

- Deep learning models are all about automatically finding appropriate transformations of the data that produce useful representations of some data.



Matrix-Matrix Multiplication

- Let us now introduce matrix-matrix multiplication of two matrices, $\boldsymbol{W}$ and $\boldsymbol{X}$:

$$\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{w}_0^T \\ \boldsymbol{w}_1^T \\ \vdots \\ \boldsymbol{w}_m^T \end{pmatrix} (\boldsymbol{x}_0 \quad \boldsymbol{x}_1 \quad \dots \quad \boldsymbol{x}_p)$$

$$= \begin{pmatrix} \boldsymbol{w}_0 \cdot \boldsymbol{x}_0 & \boldsymbol{w}_0 \cdot \boldsymbol{x}_1 & \dots & \boldsymbol{w}_0 \cdot \boldsymbol{x}_p \\ \boldsymbol{w}_1 \cdot \boldsymbol{x}_0 & \boldsymbol{w}_1 \cdot \boldsymbol{x}_1 & \dots & \boldsymbol{w}_1 \cdot \boldsymbol{x}_p \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_m \cdot \boldsymbol{x}_0 & \boldsymbol{w}_m \cdot \boldsymbol{x}_1 & \dots & \boldsymbol{w}_m \cdot \boldsymbol{x}_p \end{pmatrix}$$

Or you can write $\boldsymbol{w}_i^T \boldsymbol{x}_j$ here if $\boldsymbol{w}_i^T$ is viewed as a matrix with a single row.

Multiple Neurons and Inputs

- The matrix $\boldsymbol{X}$ represents $p + 1$ input examples, each consisting of $n + 1$ feature values, so that the number of columns in $\boldsymbol{W}$ matches the number of rows in $\boldsymbol{X}$.

- Ex 3. $\boldsymbol{W} = \begin{pmatrix} 0.9 & -0.6 & -0.5 \\ 0.2 & 0.6 & 0.6 \end{pmatrix}$,

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{pmatrix}, \boldsymbol{WX} = ?$$

We have $n + 1$ perceptrons, each having $n + 1$ inputs.

Summary

Table 1-4 Combinations of Perceptrons and Input Examples and the Corresponding Linear Algebra Operation

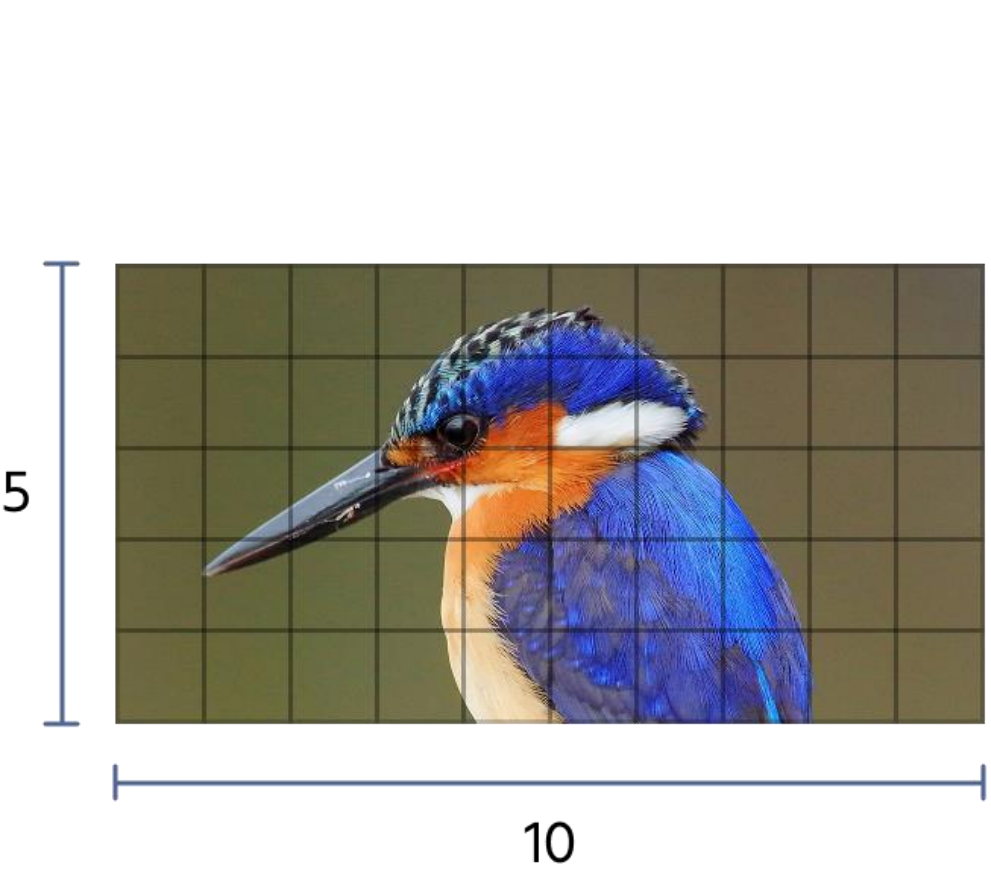
NUMBER OF PERCEPTRONS	NUMBER OF INPUT EXAMPLES	LINEAR ALGEBRA OPERATION
One	One	Dot product
Multiple	One	Matrix-vector multiplication
Multiple	Multiple	Matrix-matrix multiplication

Tensor

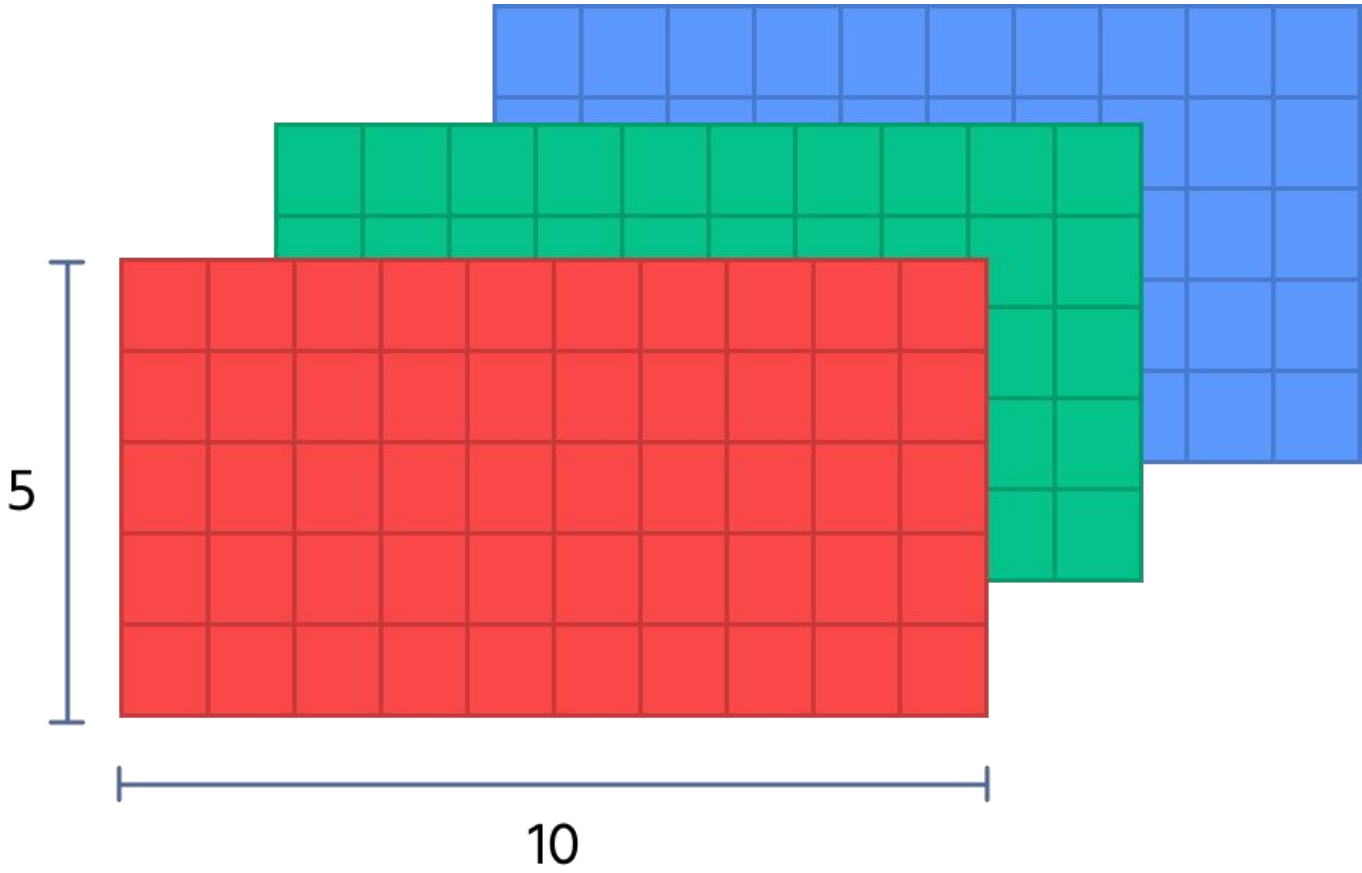
- Vectors and matrices are special cases of the more generalized concept of a tensor, which is equivalent to a multidimensional array.
- If we extended a matrix to another dimension, we would call the resulting entity a 3D tensor, such as in the case of a color image.
- The greatest challenge of calculating tensor products is to keep track of all the indices correctly.

A 3D matrix is different from a 3D tensor

2D Color Image = 3D Tensor



Original Color Image



3D RGB Tensor

Rank

- All current ML systems use tensors as their basic data structure. At its core, a tensor is a container for data—usually numerical data.

Chollet Ch.2

- The number of axes of a tensor is called its *rank*. A tensor that contains only one number is called a scalar (or rank-0 tensor, or 0D tensor).

```
Import numpy as np
x=np.array(2330)
x.ndim
```

Rank

- A rank-1 tensor has exactly one axis. Don't confuse a 5-dimensional (5D) vector with a 5D tensor!

  ```
  x=np.array([20, 23, 30, 32, 33])
  x.ndim
  ```

- What is the rank of a matrix? A matrix is a __-D tensor. A tensor is defined by three key attributes: its rank (number of axes), shape, and data type:

  ```
  x=np.array([[20, 23, 30], [23, 30, 32], [30, 32, 33]])
  x.shape
  x.dtype
  ```

Tensor Product

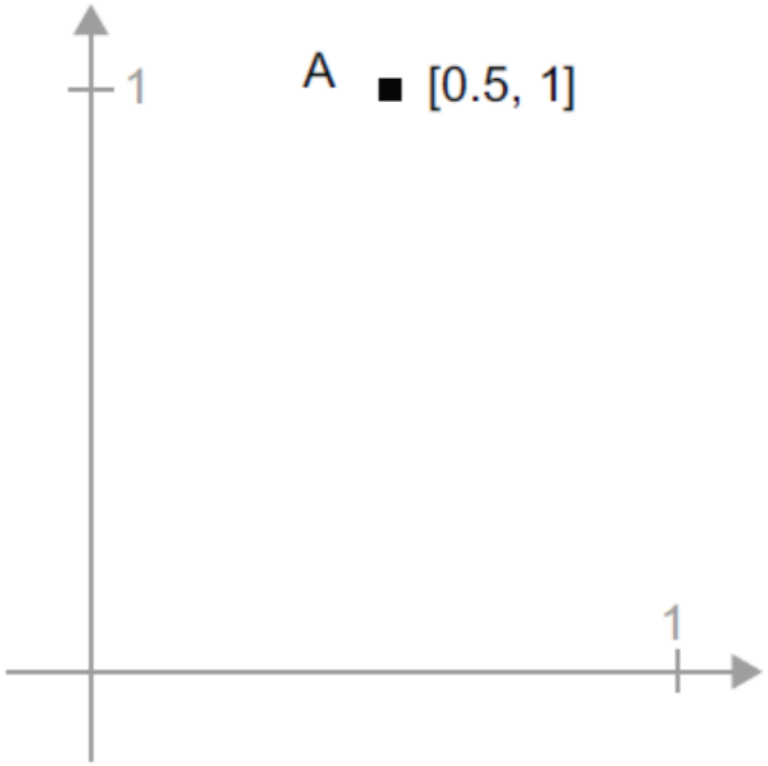
- The *tensor product*, also called *dot product* or *matmul* is one of the most common, most useful tensor operations.

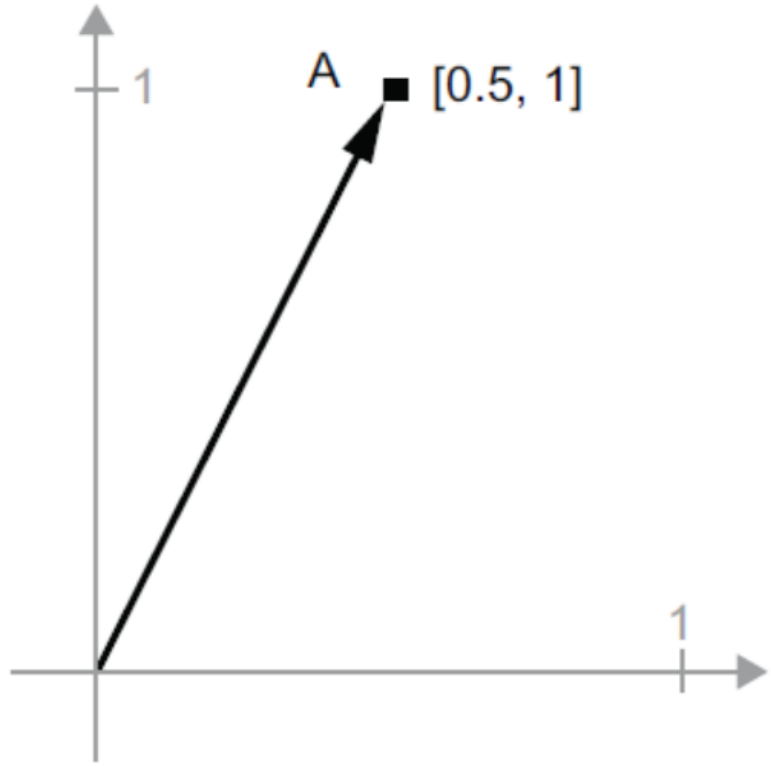
  z = np.matmul(W, X)
  z = W @ X

- You can take the product of two matrices if and only if W.shape[1] == X.shape[0]. Note that matrix multiplication may not be symmetric: matmul(W, X) isn't the same as matmul(X, W).

Geometric Interpretation

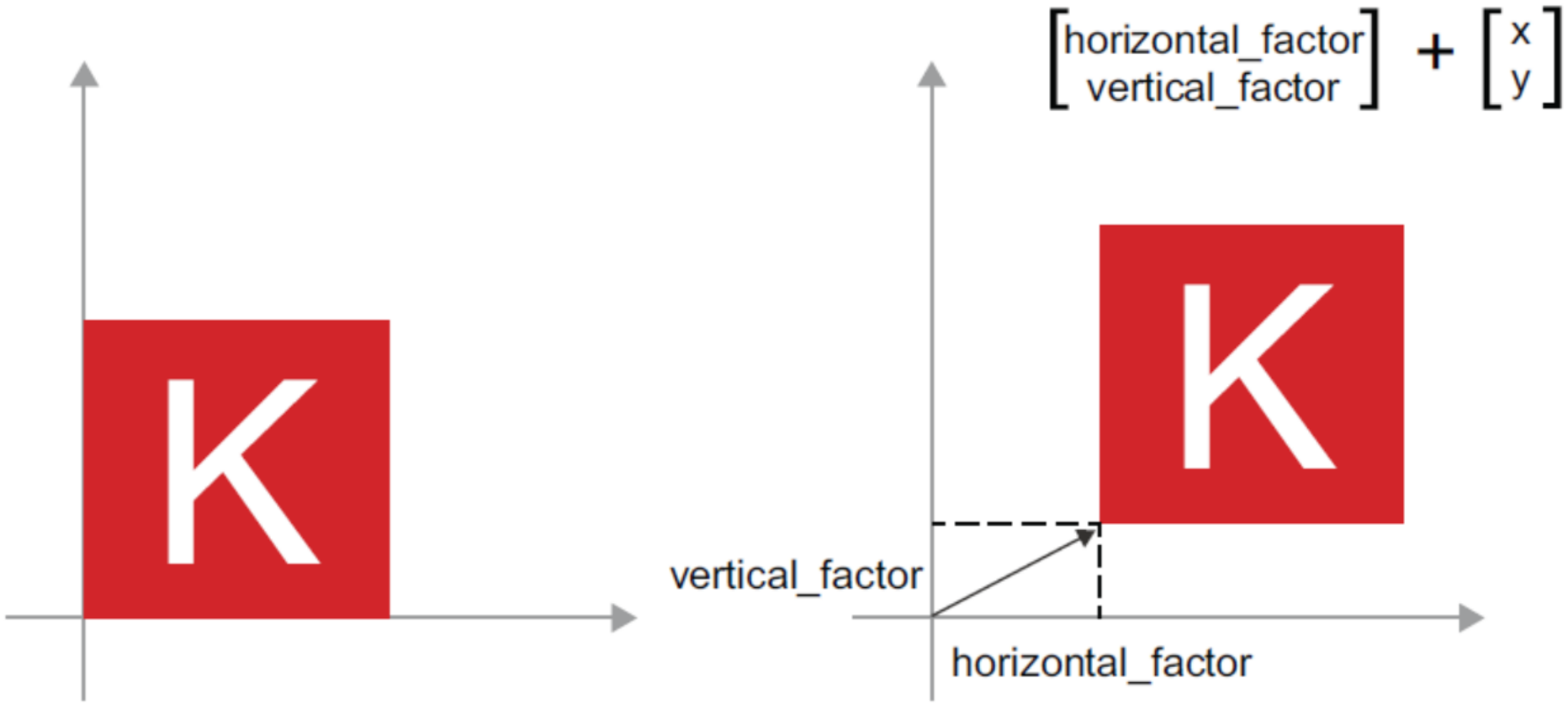
- All tensor operations have a geometric interpretation. Start with a vector A=[0.5, 1] which can be pictured as a point/an arrow in a 2D space:





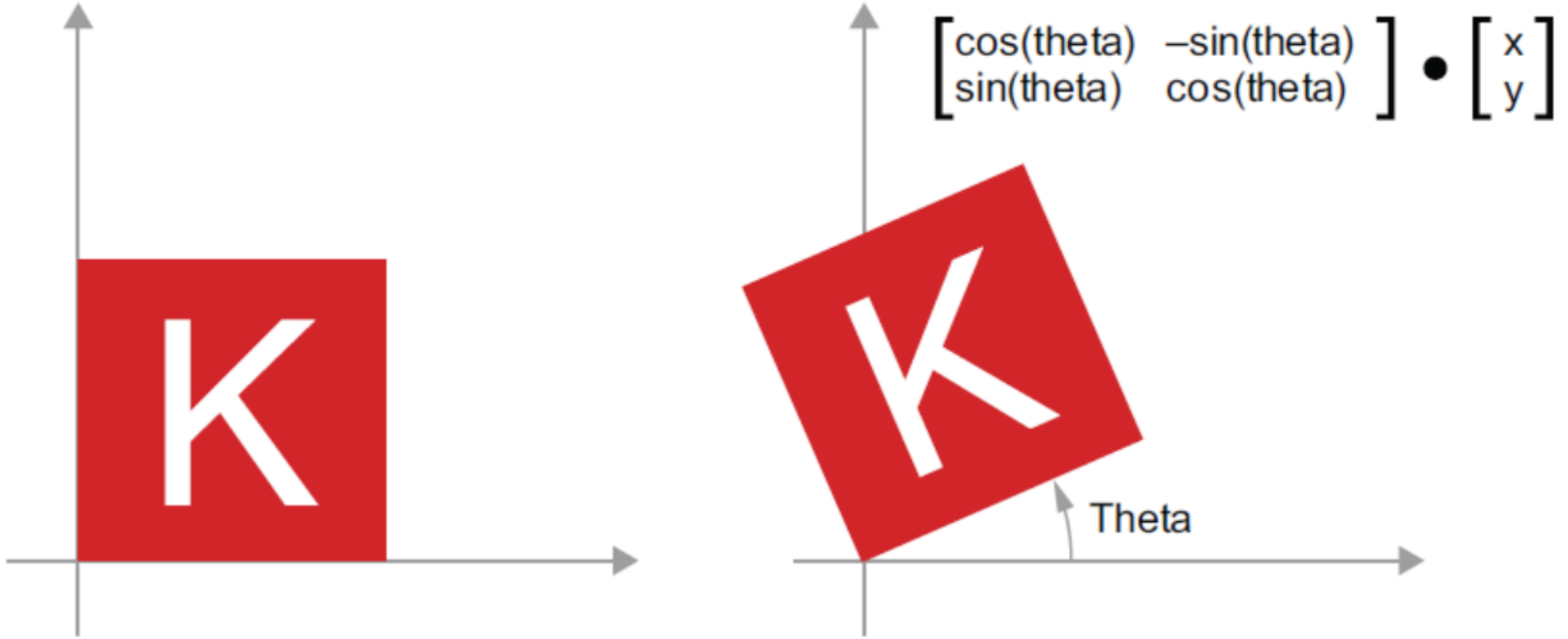
Translation

- Adding a vector to a set of points will move them by a fixed amount in a fixed direction, called a “translation.”



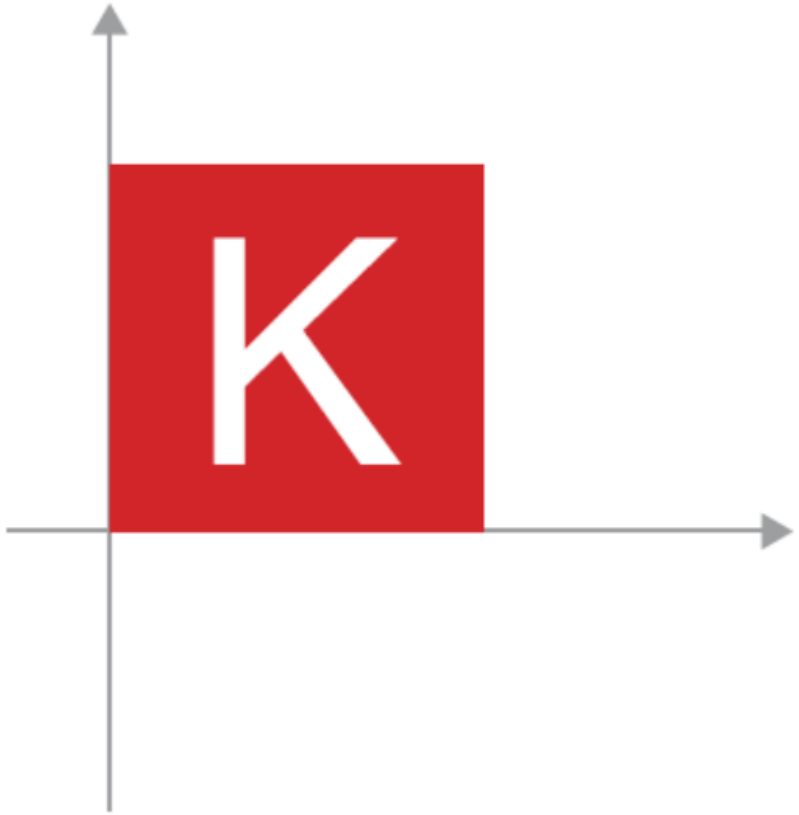
Rotation

- A counterclockwise rotation of a 2D vector by an angle Theta can be achieved via a product with a rotation matrix.

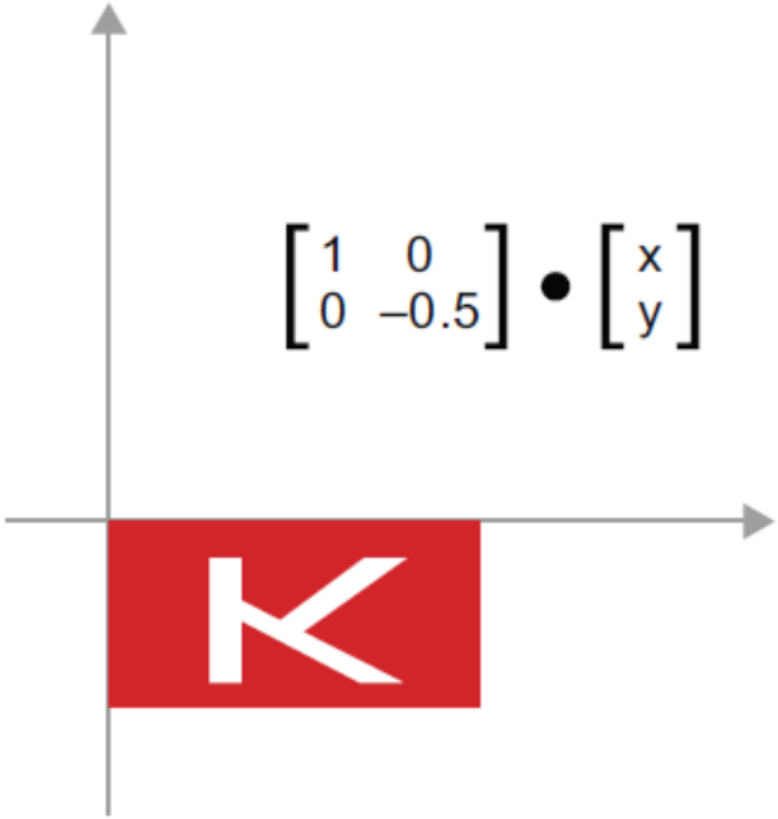


Scaling

- A vertical and horizontal scaling of the image can be achieved via a product with a diagonal matrix, which only has non-zero coefficients in its diagonal.



$$\begin{bmatrix} 1 & 0 \\ 0 & -0.5 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix}$$



Affine Transform

- You can interpret a neural network as a very complex geometric transformation in a high-dimensional space, implemented via a series of simple steps.