Electrical and Computer Engineering Department
Tarbiat Modares University

Linear Algebra

Foad Ghaderi, PhD

Linear Algebra

Linear algebra is a branch of mathematics that is widely used throughout science and engineering.

A good understanding of linear algebra is essential for understanding and working with many machine learning algorithms, especially deep learning algorithms.

Because linear algebra is a form of continuous rather than discrete mathematics, many computer scientists have little experience with it.

Scalars

- A scalar is a single number
- Integers, real numbers, rational numbers, etc.
- We denote it with *italic* font:

$$a, n, x$$

Examples:

Let $s \in \mathbb{R}$ be the slope of the line

Let $n \in \mathbb{N}$ be the number of units

Vectors

- A vector is a 1D array of numbers arranged in order.
- Can be real, binary, integer, etc.
- We denote vectors with lowercase names in *italic* **bold** typeface :

$$\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{p}$$

Examples:

$$\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Matrices

- A matrix is a 2-D array of numbers.
- Can be real, binary, integer, etc.
- We usually give matrices uppercase variable names with *italic* **bold** typeface

$$\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{P}$$

Example:

$$\boldsymbol{A} \in \mathbb{R}^{m \times n}, \quad \boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix} \quad \boldsymbol{A}_{2,:}$$

$$\boldsymbol{A}_{:,1}$$

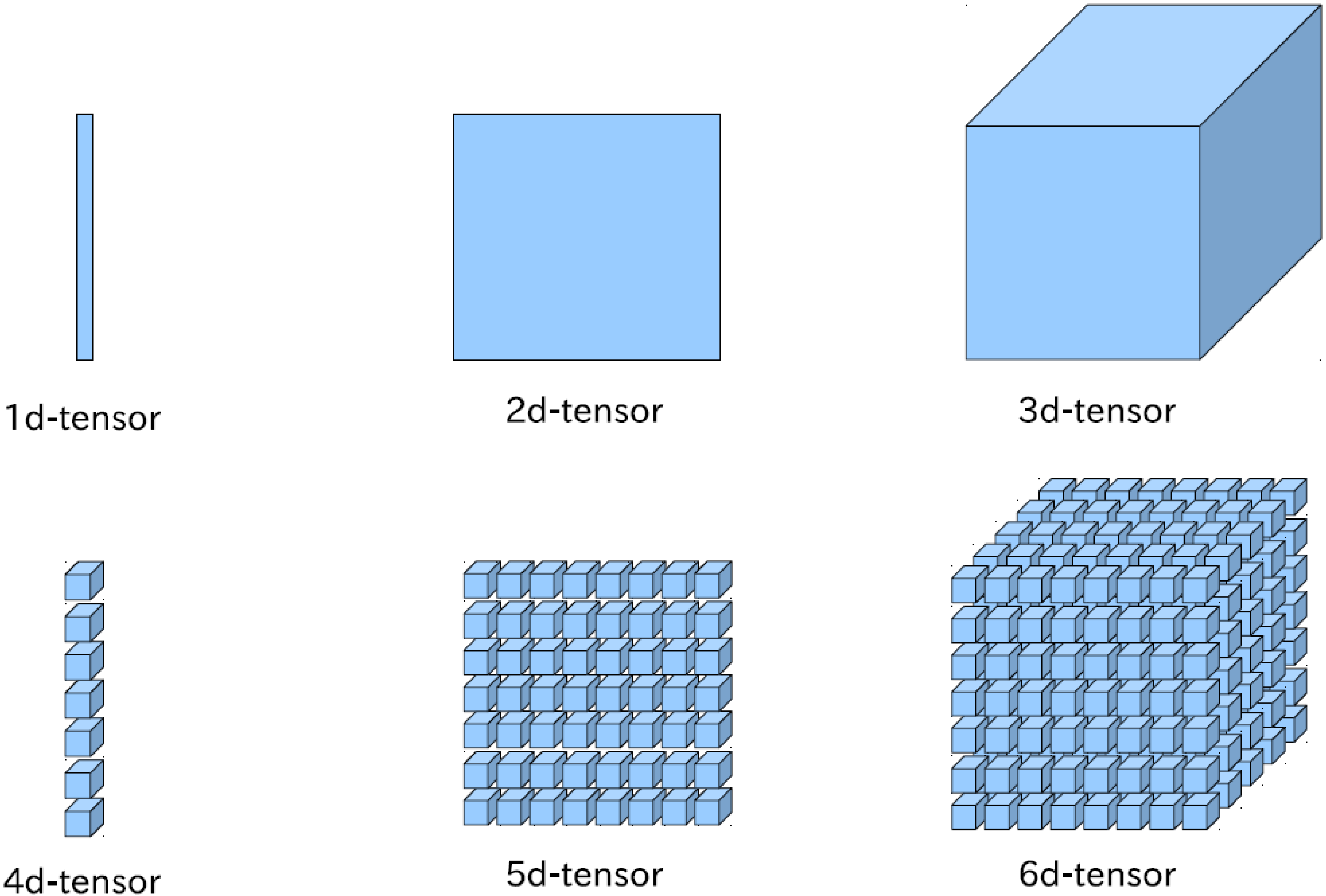
Tensors

A tensor is an array of numbers, that may have

- zero dimensions, and be a scalar
- one dimension, and be a vector
- two dimensions, and be a matrix
- or more dimensions.

Tensors

A generalization of matrices to an arbitrary number of dimensions.



Tensors

Scalars (0D tensors)

```
import numpy as np
x = np.array(10)
print(x)
print(x.ndim)
```

```
10
0
```

Vectors (1D tensors)

```
y = np.array([2, -3, 3.4, -0.03])
print(y.ndim)
y
```

```
1

array([ 2.  , -3.  ,  3.4 , -0.03])
```

Matrices (2D tensors)

```
z = np.array([[.2, -1.4],
              [1, 3.1],
              [0.4, -2]])
print(z.ndim)
print(z.shape)
z
```

```
2
(3, 2)

array([[ 0.2, -1.4],
       [ 1. ,  3.1],
       [ 0.4, -2. ]])
```

3D tensors

```
t = np.array([[[3, 2.1],
               [4, -6.7],
               [.06,1.9]],
              [[-2.05, 8.01],
               [6.4, 8.02],
               [2, 1.05]],
              [[2.05, -6.3],
               [1.08, 2.6],
               [-.08, 3.04]]])
print(t.shape)
print(t.ndim)
print(t)
```

```
(3, 3, 2)
3
[[[ 3.    2.1 ]
  [ 4.   -6.7 ]
  [ 0.06  1.9 ]]

 [[-2.05  8.01]
  [ 6.4   8.02]
  [ 2.    1.05]]

 [[ 2.05 -6.3 ]
  [ 1.08  2.6 ]
  [-0.08  3.04]]]
```

Tensors

Manipulating tensors in Numpy

Tensor slicing: selecting specific elements in a tensor

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print('Dimensionality of train images:' , train_images.shape,
      '\nDimensionality of test images:', test_images.shape)
```

```
Dimensionality of train images: (60000, 28, 28)
Dimensionality of test images: (10000, 28, 28)
```

```
my_slice = train_images[10:100]
print(my_slice.shape)
```

```
(90, 28, 28)
```

```
my_slice = train_images[10:100, :, :]
my_slice = train_images[10:100, 0:28, 0:28]
```

More examples:

```
my_slice = train_images[:, 14:, 14:]
```

14×14 pixels in the bottom-right corner of all images

```
my_slice = train_images[:, 7:-7, 7:-7]
```

14×14 pixels centered in the middle of all images

Matrix Transpose

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

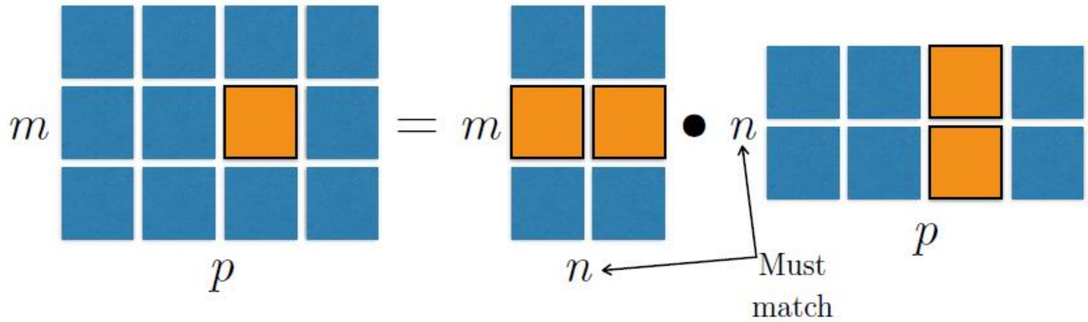
$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

$$(AB)^T = B^T A^T$$

Multiplying Matrices and Vectors

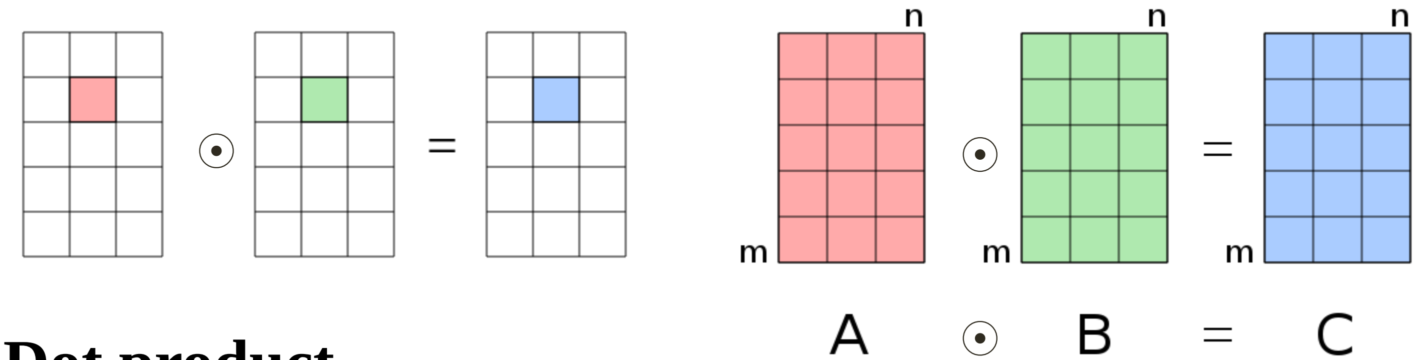
Matrix product

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



Hadamard (element-wise) product

$$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$$



Dot product

$$z = \boldsymbol{x}^T \boldsymbol{y}$$

Identity and Inverse Matrices

Identity Matrix $$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x}$$

Matrix inverse

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n$$

Matrix can’t be inverted if…

- More rows than columns
- More columns than rows
- Redundant rows/columns (“linearly dependent”, “low rank”)

Systems of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$
$$\boldsymbol{A}_{1,:}\boldsymbol{x} = \boldsymbol{b}_1$$
$$\boldsymbol{A}_{2,:}\boldsymbol{x} = \boldsymbol{b}_2$$
$$\ldots$$
$$\boldsymbol{A}_{m,:}\boldsymbol{x} = \boldsymbol{b}_m$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \rightarrow \boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = A^{-1}\boldsymbol{b} \rightarrow \boldsymbol{I}_{\boldsymbol{n}}\boldsymbol{x} = A^{-1}\boldsymbol{b} \rightarrow \boldsymbol{x} = A^{-1}\boldsymbol{b}$$

Numerically unstable, but useful for abstract analysis

A linear system of equations can have:

- No solution
- Many solutions
- Exactly one solution: this means multiplication by the matrix is an invertible function

Norms

In machine learning, we usually measure the size of vectors using a function called a *norm*.
Formally, the L^p norm is given by

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

Norms are functions mapping vectors to non-negative values, satisfying the following properties:

- $f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = \mathbf{0}$
- $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ (the *triangle inequality*)
- $\forall \alpha \in \mathbb{R}, f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$

Norms

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

- Most popular norm is L^2 norm (*Euclidean norm*), p = 2, $\|\boldsymbol{x}\|_2 = \boldsymbol{x}^T\boldsymbol{x}$ (increases very slowly near the origin)
- L^1 norm, p=1, $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$ (grows at the same rate in all locations)
- L^0 norm, number of nonzero elements.
- Max norm, infinite *p*, $L^\infty = \|\boldsymbol{x}\|_\infty = \max_i |x_i|$

Frobenius norm

The most common way to measure the size of a matrix.

$$||A||_F = \sqrt{\sum_{i,j} A_{i,j}^2}$$

Special Kinds of Matrices and Vectors

***Diagonal* matrix**:

Matrix $\boldsymbol{D}$ is diagonal if and only if $D_{i,j} = 0$ for all $i \neq j$.

Unit vector:

A vector with *unit norm* $\|\boldsymbol{x}\|_2 = 1$

***Symmetric* matrix**:

Any matrix that is equal to its own transpose, $\boldsymbol{A} = \boldsymbol{A}^T$

Orthogonal vectors:

A vector $\boldsymbol{x}$ and a vector $\boldsymbol{y}$ are *orthogonal* to each other if $\boldsymbol{x}^T\boldsymbol{y} = \boldsymbol{0}$.

If the vectors are not only orthogonal but also have unit norm, they are called ***orthonormal***.

In $\mathbb{R}^{\boldsymbol{n}}$, at most n vectors may be mutually orthogonal with nonzero norm.

Orthogonal matrix

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I} \rightarrow \boldsymbol{A}^{-1} = \boldsymbol{A}^T$$

Eigendecomposition

An *eigenvector* of a square matrix A is a non-zero vector v such that multiplication by A alters only the scale of v:
$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$

Suppose that a matrix A has *n* linearly independent eigenvectors, $\{v^{(1)}, \ldots, v^{(n)}\}$, with corresponding eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$.

$$\boldsymbol{V} = [v^{(1)}, \ldots, v^{(n)}]$$
$$\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]^T.$$

The *eigendecomposition* of $\boldsymbol{A}$ is then given by

$$\boldsymbol{A} = \boldsymbol{V}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}.$$

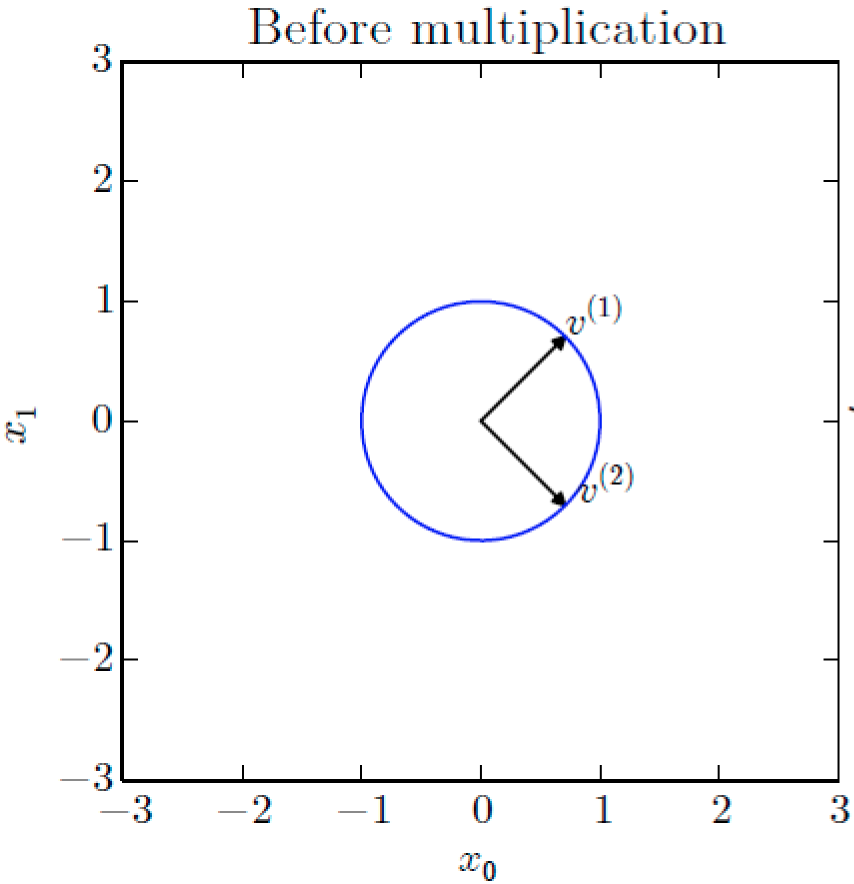
Eigendecomposition

- Not every matrix can be decomposed into eigenvalues and eigenvectors. In some cases, the decomposition exists, but may involve complex rather than real numbers.

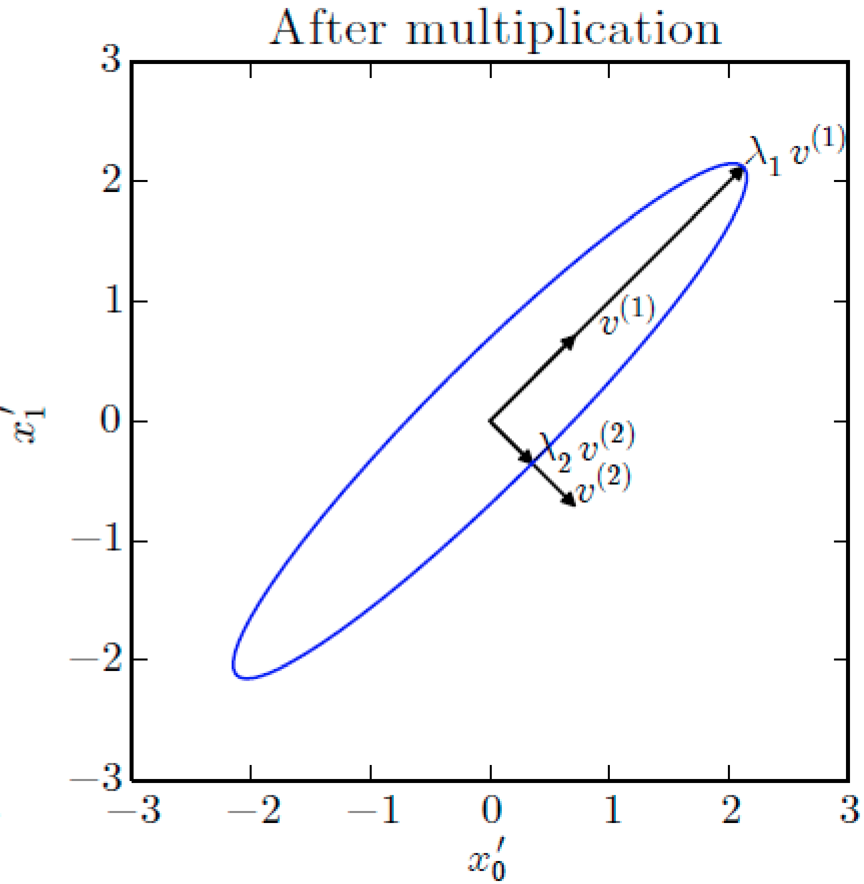
- Every real symmetric matrix can be decomposed into an expression using only real-valued eigenvectors and eigenvalues:
$$\boldsymbol{A} = \boldsymbol{Q\Lambda Q}^T,$$
where $\boldsymbol{Q}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix.

Eigendecomposition

An example of the effect of eigenvectors and eigenvalues.



We plot the set of all unit vectors $\boldsymbol{u} \in \mathbb{R}^{\mathbf{2}}$ as a unit circle.



We plot the set of all points $\boldsymbol{Au}$

Singular Value Decomposition

Singular Value Decomposition

Suppose that $\boldsymbol{A}$ is an $m \times n$ matrix

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}$$

where $\boldsymbol{U}$ is an $m \times m$ matrix, $\boldsymbol{D}$ is an $m \times n$ matrix, and $\boldsymbol{V}$ is an $n \times n$ matrix. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are both defined to be orthogonal matrices.

The matrix $\boldsymbol{D}$ is defined to be a diagonal (not necessarily square) matrix. The elements along the diagonal of $\boldsymbol{D}$ are known as the *singular values* of the matrix $\boldsymbol{A}$. The columns of $\boldsymbol{U}$ are known as the *left-singular vectors*. The columns of $\boldsymbol{V}$ are known as the *right-singular vectors*.

- The SVD is more generally applicable than EVD.
- Every real matrix has a singular value decomposition, but the same is not true for the eigenvalue decomposition.
- If a matrix is not square, the eigendecomposition is not defined

- If a matrix is not square, the eigendecomposition is not defined

Neural networks and deep learning, Spring 2020 20

Singular Value Decomposition

Suppose that $\boldsymbol{A}$ is an $m \times n$ matrix

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}$$

Interpretation of the SVD in terms of EVD:

- The left-singular vectors of A are the eigenvectors of AA^T.
- The right-singular vectors of A are the eigenvectors of A^TA.
- The non-zero singular values of A are the square roots of the eigenvalues of A^TA.

The Moore-Penrose Pseudoinverse

Matrix inversion is not defined for matrices that are not square. Find B as the left-inverse of a matrix A, so that we can solve a linear equation

$$\boldsymbol{Ax = y}$$

by left-multiplying each side to obtain

$$\boldsymbol{x = By}.$$

If A is taller than it is wide, possibly no solution
If A is wider than it is tall, possibly multiple solutions.

The Moore-Penrose Pseudoinverse

The SVD allows the computation of the pseudoinverse:

$$\boldsymbol{A}^{+} = \boldsymbol{V}\,\boldsymbol{D}^{+}\boldsymbol{U}^{T}$$

$\boldsymbol{U}$, $\boldsymbol{D}$ and $\boldsymbol{V}$ are the singular value decomposition of $\boldsymbol{A}$, and the pseudoinverse $\boldsymbol{D}^{+}$ of a diagonal matrix $\boldsymbol{D}$ is obtained by

- taking the reciprocal of its non-zero elements
- then taking the transpose of the resulting matrix.

The Moore-Penrose Pseudoinverse

The SVD allows the computation of the pseudoinverse:

$$\boldsymbol{A}^{+} = \boldsymbol{V}\,\boldsymbol{D}^{+}\boldsymbol{U}^{T}$$

- When $\boldsymbol{A}$ has more columns than rows, then solving a linear equation using the pseudoinverse provides the solution $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{y}$ with minimal Euclidean norm $||\boldsymbol{x}||_2$ among all possible solutions.
- When $\boldsymbol{A}$ has more rows than columns, it is possible for there to be no solution. In this case, using the pseudoinverse gives us the $\boldsymbol{x}$ for which $\boldsymbol{Ax}$ is as close as possible to $\boldsymbol{y}$ in terms of Euclidean norm $||\boldsymbol{Ax} - \boldsymbol{y}||_2$.

Trace

$$Tr(\boldsymbol{A}) = \sum_i \boldsymbol{A}_{i,i}$$

$$||\boldsymbol{A}||_F = \sqrt{Tr(\boldsymbol{A}\boldsymbol{A}^T)}$$

$$Tr(\boldsymbol{A}) = Tr(\boldsymbol{A}^T)$$

$$Tr(\boldsymbol{ABC}) = Tr(\boldsymbol{CAB}) = Tr(\boldsymbol{BCA})$$