Data Representation

Graphs and Matrices

Tirtharaj Dash

BITS Pilani, K.K. Birla Goa Campus

tirtharaj@goa.bits-pilani.ac.in

September 12, 2020

Graphs I

Many real-world relationships can be visualized as a graph.

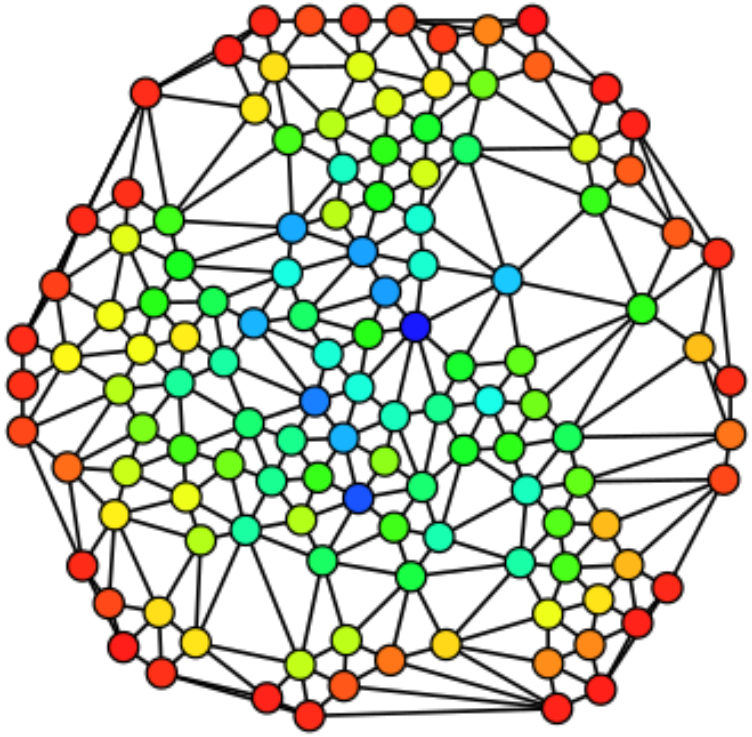


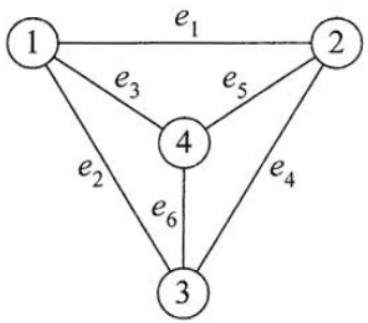
Figure: An example of a graph (Source: Wikipedia)

Graphs II

Graph

A graph $G = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges.

A vertex is also called a node. An edge is also called a link. Let's look at the following example:



Here, $V = \{1, 2, 3, 4\}$ and $E = \{e_1, e_2, \ldots, e_6\}$.

Graphs III

Order and size

The order of a graph is $|V|$, and its size is $|E|$.

The order of the graph in the previous example is 4 and size is 6.

Edge

If u and v are two vertices of a graph and if the unordered pair $\{u, v\}$ is an edge denoted by e, we say that e joins u and v or that it is an edge between u and v. The vertices u and v are said to be incident on e and e is incident to both u and v.

Example: $e_1 = \{1, 2\}$, $e_3 = \{1, 4\}$.

Graphs IV

Parallel edge

Two or more edges that join the same pair of distinct vertices are called parallel edge.

Loop

An edge represented by an unordered pair in which the two elements are not distinct is known as a loop.

Multigraph

A graph with no loops is a multigraph.

Pseudograph

A graph with at least one loop is a pseudograph.

Graphs V

Simple graph

A simple graph is a graph with no parallel edges and loops.

Complete graph

The complete graph K_n is a graph with n vertices in which there is exactly one edge joining every pair of vertices.

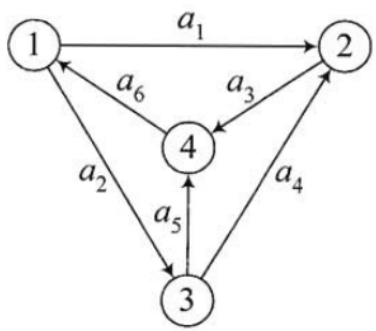
The graph K_1 with one vertex and no edge is known as the **trivial** graph.

Graphs VI

Directed graph

A directed graph or digraph consists of a finite set V of vertices and a set A of ordered pairs of distinct vertices called arcs. If the ordered pair u, v is an arc a, we say that the arc a is directed from u to v. In this context, arc a is adjacent from vertex u and is adjacent to vertex v.

The following is a directed graph:



Graphs VII

Degree of a vertex

In a graph with no loops, the degree of a vertex is the number of edges adjacent to that vertex. In a graph with no loops, a vertex is said to be an isolated vertex if its degree is 0 and an end- vertex if its degree is 1.

Degree of a vertex in a diagraph

In a digraph, the number of arcs adjacent to a vertex is the indegree of that vertex, and the number of arcs adjacent from a vertex is the outdegree of that vertex.

Graphs VIII

We will define an adjacency matrix for a graph with no self loop.

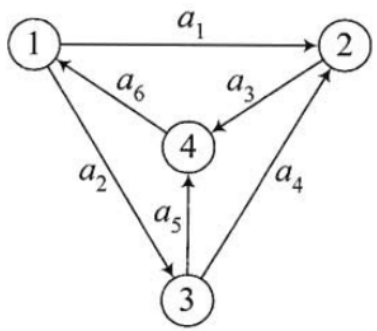
Adjacency matrix

Let $G = (V, E)$ be a graph where $V = 1, 2, \ldots, n$. The adjacency matrix of the graph is the $n \times n$ matrix $A = [a_{ij}]$, where the nondiagonal entry a_{ij} is the number of edges joining vertex i and vertex j.

- A graph with no self loop will have diagonal elements of A as 0s.
- The adjacency matrix of a simple graph is a binary matrix (0, 1 matrix) in which each diagonal entry is zero.
- The adjacency matrix of the complete graph K_n, each nondiagonal entry is 1.

Graphs IX

For the following graph:



the adjacency matrix is

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Matrices I

Scalars A scalar is just a single number.

Example 1

$x = 3.5$. Here, x is a scalar; $x \in \mathbb{R}$.

Example 2

Similarly, Iris dataset has $d = 4$ features. Here, $d \in \mathbb{N}$ is a scalar.

Sometimes, we write a scalar as $(a)_{1\times 1}$.

Matrices II

Vectors A vector is an array of number.

Example

$\mathbf{x} = [1.3, -4.0, 11.7, 4]^{\mathsf{T}}$ is a vector; $\mathbf{x} \in \mathbb{R}^4$.

A d-dimensional vector is written as $\mathbf{x} \in \mathbb{R}^d$ and in matrix form as: $(\cdots)_{d\times 1}$.

In a programming convention, we refer $(\cdots)_{d\times 1}$ as an 1-D array with d-elements.

Matrices III

Matrices A matrix is a 2-D array of numbers.

Example

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 6 \\ 3 & 6 & 8 \end{bmatrix} \text{ is a } 3 \times 3 \text{ matrix.}$$

We write a matrix as $\mathbf{A} \in \mathbb{R}^{m \times n}$.

We read this as: $\mathbf{A}$ is a real-valued matrix of order $m \times n$, where m is number of rows, n is number of columns.

An element of a matrix is referenced as $a_{i,j} \in \mathbf{A}$, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Matrices IV

- Addition and subtraction of matrices are additions and subtractions of corresponding entries. For example:

$$\mathbf{M} \pm \mathbf{N} = \begin{bmatrix} m_{11} \pm n_{11} & m_{12} \pm n_{12} \\ m_{21} \pm n_{21} & m_{22} \pm n_{22} \end{bmatrix}$$

(and similarly for higher dimension matrices)

- Multiplication of an $m \times n$ matrix by an $n \times k$ matrix is a $m \times k$ matrix consisting of "sums of row-column products". For example:

$$\mathbf{MN} = \begin{bmatrix} m_{11}n_{11} + m_{12}n_{21} & m_{11}n_{12} + m_{12}n_{22} \\ m_{21}n_{11} + m_{22}n_{21} & m_{21}n_{12} + m_{22}n_{22} \end{bmatrix}$$

That is, entry 11 of the product is the sum of products from Row 1, Col 1; entry 12 is the sum of products from Row 1, Col 2 and so on.

Matrices V

- New vectors (matrices) can be obtained from old ones by linear combination of the components. For example, the new vector **y** with components:

$$\begin{aligned} y_1 &= m_{11}x_1 + m_{12}x_2 \\ y_2 &= m_{21}x_1 + m_{22}x_2 \end{aligned}$$

is a *linear transformation* of the vector **x** and can be written in matrix notation as:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

or:

$$\mathbf{y} = \mathbf{M}\mathbf{x}$$

This kind of transformation is an example of a *rotational* transformation

Matrices VI

- The *inverse* of a matrix $\mathbf{M}$ is the matrix $\mathbf{M}^{-1}$ s.t.

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$$

  where $\mathbf{I}$ is the *identity* matrix (a matrix with 1's along the diagonal, and 0's everywhere else). The inverse of a matrix may not always exist, and is not always easy to compute. But it will usually be done by a program, rather than by hand.

- Calculation of the inverse requires knowledge of the determinant $|\mathbf{M}|$ or $det(\mathbf{M})$. In 2 dimensions:

$$|\mathbf{M}| = det(\mathbf{M}) = \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} = m_{11}m_{22} - m_{12}m_{21}$$

- Division of matrices is not defined: the inverse of a matrix takes the place of division

Matrices VII

- Multiplication of a vector $\mathbf{x}$ by a matrix $\mathbf{M}$ transforms $\mathbf{x}$ to a vector $\mathbf{y}$. Sometimes, the same effect can be achieved by multiplying $\mathbf{x}$ by a number (which may be a complex number) λ. That is:

$$\mathbf{Mx} = \lambda\mathbf{x}$$

or:

$$(\mathbf{M} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$$

- It can be shown that for the above to hold, either $\mathbf{x} = \mathbf{0}$ or $det(\mathbf{M} - \lambda\mathbf{I}) = 0$. This is a polynomial equation in λ. The solutions of λ for which $det(\mathbf{M} - \lambda\mathbf{I}) = 0$ are the *eigenvalues* of $\mathbf{M}$. The resulting values of $\mathbf{x}$ that satisfy $\mathbf{Mx} = \lambda\mathbf{x}$ are called the *eigenvectors* of $\mathbf{M}$

Matrices VIII

- The transpose of a matrix $\mathbf{M}$, denoted by $\mathbf{M}^T$ is obtained by "rotating" the matrix. That is, rows are made into columns. For example:

$$\text{if } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ then } \mathbf{x}^T = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

- If:

$$\mathbf{M}^T = \mathbf{M}$$

then $\mathbf{M}$ is said to be symmetric

Matrices IX

- Given a vector $\mathbf{x}$, the operation $\mathbf{x}\mathbf{x}^T$ results in in a matrix. For example:

$$\mathbf{x}\mathbf{x}^T = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 x_2 \\ x_1 x_2 & x_2^2 \end{bmatrix}$$

But the operation $\mathbf{x}^T\mathbf{x}$ results in a number (scalar):

$$\mathbf{x}^T\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + x_2^2$$

In general, provided the vectors are compatible, $\mathbf{x}\mathbf{y}^T$ is a matrix and $\mathbf{x}^T\mathbf{y}$ is a scalar. The latter is also called the *dot* or *inner* product $\mathbf{x} \cdot \mathbf{y}$ of the vectors. If the dot product of a pair of vectors is 0, then the vectors are said to be *orthogonal*

Matrices X

- An *orthogonal* matrix M is a $n \times n$ matrix whose rows and columns are orthogonal. That is:

$$\mathbf{M}^T\mathbf{M} = \mathbf{M}\mathbf{M}^T = \mathbf{I}$$

or:

$$\mathbf{M}^T = \mathbf{M}^{-1}$$

- Let $\mathbf{M}$ be a matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. Then

$$\mathbf{M}\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

Matrices XI

Tensors In some cases we will need an array with more than two axes (dimensions). A tensor is an array of numbers arranged on a regular grid with a variable number of axes.

Example

A photo that you clicked is stored as a tensor: (3-depth; RGB) or (4-depth; CMYK).

In an image, the first two axes are *height* and *width* of the image and the third axis refers to *depth* (no. of color channels).

Norms I

- Norm measures the size of a vector.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

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

for $p \in \mathbb{R}$, $p \geq 1$.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

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

  for $p \in \mathbb{R}$, $p \geq 1$.
- Norm is a function that maps a vector to a non-negative value.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

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

  for $p \in \mathbb{R}$, $p \geq 1$.
- Norm is a function that maps a vector to a non-negative value.
- Intuitively, norm measures the distance of a vector $\mathbf{x}$ from origin.

Norms II

Norm is any function f that satisfies the following properties:

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

Norms II

Norm is any function f that satisfies the following properties:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$
- $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (*triangle inequality*)

Norms II

Norm is any function f that satisfies the following properties:

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

Norms III

Euclidean norm Most frequently, we will be using L^2 norm in ML. It is enoted as $||\mathbf{x}||$.

$$||\mathbf{x}|| = \mathbf{x}^T\mathbf{x}$$

L^1 **norm** There will be situations in ML where difference between a zero and a non-zero element is very important. In such cases, we will use L^1 norm.

$$||\mathbf{x}||_1 = \sum_i |x_i|$$

i.e. every time an element x_i moves ϵ-away from 0, the norm increases by ϵ.

Max norm It is called L^∞ norm, which is calculated as

$$||\mathbf{x}||_\infty = \max_i |x_i|$$

Norms IV

Frobenius norm It measures the size of a matrix.

This norm is used most frequently in deep learning.

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

Relationship between dot product and norm The dot product of two vectors $\mathbf{x}$ and $\mathbf{y}$ can be written as

$$\mathbf{x}^\mathsf{T}\mathbf{y} = ||\mathbf{x}||_2 ||\mathbf{y}||_2 \cos\theta$$

where, θ is angle between the two vectors.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.
- Deep Learning is a problem for which SIMD is well-suited.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.
- Deep Learning is a problem for which SIMD is well-suited.
- Example: You want to compute non-linear activation of a matrix:
  1. Either you can call the `transform()` operation for each element. Total $m \times n$ calls.
  2. Or, call the `transform()` operation once for whole matrix.

Parallelism II

Given a matrix $\mathbf{Z}$, compute its non-linear activation $\sigma(\mathbf{Z})$.

1. Using explicit for loop:

```
A = np.zeros(Z.shape)
for i in range(0, Z.shape[0]):
    for j in range(0, Z.shape[1]):
        A[i][j] = 1 / (1 + np.exp(-Z[i][j]))
```

2. Using **vectorisation**:

```
A = 1 / (1 + np.exp(-Z))
```

Vectorisation I

Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$, $b \in R$. We want to compute $z = \mathbf{w}^\mathsf{T}\mathbf{x} + b$.

1. Without vectorisation

```
z = 0.
for i in range(d):
    z += W[i] * X[i]
z += b
```

2. With vectorisation

```
z = np.dot(W, X) + b
```

Vectorisation II

Computation time (64GB RAM, 16-core Intel Xeon CPU, 3.10GHz):

d	non-vec(s)	vec(s)
10^3	6.1×10^{-4}	2.6×10^{-5}
10^6	0.622	0.003
10^7	6.099	0.008
10^8	55.191	0.081
10^9	–	0.487

– : couldn't wait!

Vectorisation III

Vectorised matrix multiplication ($A, B \in \mathbb{R}^{d \times d}$). Same machine with 8GB GPGPU. Comparing CPU and GPU:

d	CPU(s)	GPU(s)
10^4	5.600	0.001

*Both results are obtained using PyTorch.