Linear Algebra

Lecture 4

Centre for Data Science, ITER
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.

Contents

Introduction

- Linear algebra is the branch of mathematics that deals with vector spaces.
- It underpins a large number of data science concepts and techniques.

Vectors

- Vectors are points in some finite-dimensional space.
- They are often a useful way to represent numeric data.
- The simplest approach is to represent vectors as lists of numbers.

Creating data type 'Vector'

- A list of three numbers corresponds to a vector in three dimensional space, and vice versa.
  Eg: [2,4,1] is a three dimensional vector.
- We'll accomplish this with a type alias that says a Vector is just a list of floats.

```
from typing import List
Vector = List[float]
```

Performing arithmetic on vectors

- Throughout data science we will need to perform arithmetic on vectors.
- Eg: We might need to add two vectors or we might need to subtract one vector from the other.
- We'll build these arithmetic tools ourselves because Python lists aren't vectors.

Defining Addition

- Vectors are added *component-wise*.
- This means that if two vectors v and w are the same length, their sum is just the vector whose first element is v[0] + w[0], whose second element is v[1] + w[1], and so on.
- If they're not the same length, then we're not allowed to add them.
- Eg: Adding the vectors [1, 2] and [2, 1] results in [1 + 2, 2 + 1] or [3, 3].

Defining Addition (Contd.)

- We can easily implement addition by *zip-ing* the vectors together and using a list comprehension to add the corresponding elements:

```
def add(v: Vector, w: Vector) -> Vector:
    assert len(v) == len(w)
    return [v_i + w_i for v_i, w_i in zip(v, w)]
```

Defining Subtraction

- To subtract two vectors we just subtract the corresponding elements:

```
def subtract(v: Vector, w: Vector) -> Vector:
    assert len(v) == len(w)
    return [v_i - w_i for v_i, w_i in zip(v, w)]
```

Adding more than two vectors

- If we want to component-wise sum a list of vectors—that is, create a new vector whose first element is the sum of all the first elements, whose second element is the sum of all the second elements, and so on then we can use:

```
def vector_sum(vectors:  List[Vector]) -> Vector:
    assert vectors
    num_elements = len(vectors[0])
    assert all(len(v) == num_elements for v in vectors)
    return [sum(vector[i] for vector in vectors) for i in range(num_elements)]
```

Scalar multiplication of vectors

- We'll also need to be able to multiply a vector by a scalar, which we do simply by multiplying each element of the vector by that number:

```
def scalar_multiply(c:  float, v:  Vector) -> Vector:
      return [c * v_i for v_i in v]
```

Component-wise mean

- This allows us to compute the component-wise means of a list of (same-sized) vectors:

```
def vector_mean(vectors:  List[Vector]) -> Vector:
      n = len(vectors)
      return scalar_multiply(1/n, vector_sum(vectors))
```

Dot product

- The dot product of two vectors is the sum of their component-wise products:

```
def dot(v:  Vector, w:  Vector) -> float:
     assert len(v) == len(w)
     return sum(v_i * w_i for v_i, w_i in zip(v, w))
```

Computing vector's sum of squares and magnitude

- It's easy to compute a vector's sum of squares:

```
def sum_of_squares(v:  Vector) -> float:
    return dot(v, v)
```

- We can use this function to compute magnitude (or length) of the vector:

```
import math
def magnitude(v:  Vector) -> float:     return
math.sqrt(sum_of_squares(v))
```

Distance between two vectors

- We now have all the pieces we need to compute the distance between two vectors, defined as:

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

- In code we define the squared distance as:

```
def squared_distance(v: Vector, w: Vector) -> float:
    return sum_of_squares(subtract(v, w))
```

- Now we define the distance function:

```
def distance(v: Vector, w: Vector) -> float:
    return math.sqrt(squared_distance(v, w))
```

Matrix

- A matrix is a two-dimensional collection of numbers.
- We will represent matrices as lists of lists, with each inner list having the same size and representing a row of the matrix.
- If A is a matrix, then A[i][j] is the element in the i^{th} row and the j^{th} column.
- The matrix A has len(A) rows and len(A[0]) columns, which we consider its shape:

```
from typing import Tuple
def shape(A: Matrix) -> Tuple[int, int]:
      num_rows = len(A)
      num_cols = len(A[0]) if A else 0
      return num_rows, num_cols
```

Matrix (Contd.)

- We can get the number of rows as follows:

```
def get_row(A: Matrix, i:  int) -> Vector:
    return A[i]
```

- We can get the number of columns as follows:

```
def get_column(A: Matrix, j:  int) -> Vector:
    return [A_i[j] for A_i in A]
```

Defining matrices

- We'll also want to be able to create a matrix given its shape and a function for generating its elements.

```
from typing import Callable
def make_matrix(num_rows:  int,
                num_cols:  int,
                entry_fn:  Callable[[int, int], float]) -> Matrix:
    return [[entry_fn(i, j)
             for j in range(num_cols)]
             for i in range(num_rows)]
```

Identity Matrix

- We can define identity matrix using the above function as follows:

```
def identity_matrix(n:  int) -> Matrix:
   return make_matrix(n, n, lambda i, j:  1 if i == j else 0)
```

Conclusion

- Matrices will be important to us for several reasons.
- We can use a matrix to represent a data set consisting of multiple vectors, simply by considering each vector as a row of the matrix.
- We can use an $n \times k$ matrix to represent a linear function that maps k-dimensional vectors to n-dimensional vectors.
- Matrices can be used to represent binary relationships.

References

[1] Data Science from Scratch: First Principles with Python by Joel Grus

Thank You
Any Questions?