to the class P when one does not have the luxury of an infinite number of processors or omniscient guessing. Most computer scientists are convinced that P $\neq$ NP; that NP problems are inherently hard and have no polynomial-time algorithms. But this has never been proven.

NP-COMPLETE

Those who are interested in deciding whether P = NP look at a subclass of NP called the **NP-complete** problems. The word "complete" is used here in the sense of "most extreme" and thus refers to the hardest problems in the class NP. It has been proven that either all the NP-complete problems are in P or none of them is. This makes the class theoretically interesting, but the class is also of practical interest because many important problems are known to be NP-complete. An example is the satisfiability problem: given a sentence of propositional logic, is there an assignment of truth values to the proposition symbols of the sentence that makes it true? Unless a miracle occurs and P = NP, there can be no algorithm that solves *all* satisfiability problems in polynomial time. However, AI is more interested in whether there are algorithms that perform efficiently on *typical* problems drawn from a predetermined distribution; as we saw in Chapter 7, there are algorithms such as WALKSAT that do quite well on many problems.

CO-NP

CO-NP-COMPLETE

The class **co-NP** is the complement of NP, in the sense that, for every decision problem in NP, there is a corresponding problem in co-NP with the "yes" and "no" answers reversed. We know that P is a subset of both NP and co-NP, and it is believed that there are problems in co-NP that are not in P. The **co-NP-complete** problems are the hardest problems in co-NP.

The class #P (pronounced "sharp P") is the set of counting problems corresponding to the decision problems in NP. Decision problems have a yes-or-no answer: is there a solution to this 3-SAT formula? Counting problems have an integer answer: how many solutions are there to this 3-SAT formula? In some cases, the counting problem is much harder than the decision problem. For example, deciding whether a bipartite graph has a perfect matching can be done in time $O(VE)$ (where the graph has V vertices and E edges), but the counting problem "how many perfect matches does this bipartite graph have" is #P-complete, meaning that it is hard as any problem in #P and thus at least as hard as any NP problem.

Another class is the class of PSPACE problems—those that require a polynomial amount of space, even on a nondeterministic machine. It is believed that PSPACE-hard problems are worse than NP-complete problems, although it could turn out that NP = PSPACE, just as it could turn out that P = NP.

A.2 Vectors, Matrices, and Linear Algebra

VECTOR

Mathematicians define a **vector** as a member of a vector space, but we will use a more concrete definition: a vector is an ordered sequence of values. For example, in two-dimensional space, we have vectors such as $\mathbf{x} = \langle 3, 4 \rangle$ and $\mathbf{y} = \langle 0, 2 \rangle$. We follow the convention of boldface characters for vector names, although some authors use arrows or bars over the names: $\vec{x}$ or $\bar{y}$. The elements of a vector can be accessed using subscripts: $\mathbf{z} = \langle z_1, z_2, \ldots, z_n \rangle$. One confusing point: this book is synthesizing work from many subfields, which variously call their sequences vectors, lists, or tuples, and variously use the notations $\langle 1, 2 \rangle$, [1, 2], or (1, 2).

The two fundamental operations on vectors are vector addition and scalar multiplication. The vector addition $\mathbf{x}+\mathbf{y}$ is the elementwise sum: $\mathbf{x}+\mathbf{y} = \langle 3+0, 4+2 \rangle = \langle 3, 6 \rangle$. Scalar multiplication multiplies each element by a constant: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

The length of a vector is denoted $|\mathbf{x}|$ and is computed by taking the square root of the sum of the squares of the elements: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. The dot product $\mathbf{x} \cdot \mathbf{y}$ (also called scalar product) of two vectors is the sum of the products of corresponding elements, that is, $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$, or in our particular case, $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Vectors are often interpreted as directed line segments (arrows) in an n-dimensional Euclidean space. Vector addition is then equivalent to placing the tail of one vector at the head of the other, and the dot product $\mathbf{x} \cdot \mathbf{y}$ is equal to $|\mathbf{x}|\ |\mathbf{y}|\ \cos\theta$, where θ is the angle between $\mathbf{x}$ and $\mathbf{y}$.

MATRIX

A **matrix** is a rectangular array of values arranged into rows and columns. Here is a matrix $\mathbf{A}$ of size 3×4:

$$\begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \end{pmatrix}$$

The first index of $\mathbf{A}_{i,j}$ specifies the row and the second the column. In programming languages, $\mathbf{A}_{i,j}$ is often written `A[i,j]` or `A[i][j]`.

The sum of two matrices is defined by adding their corresponding elements; for example $(\mathbf{A}+\mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$. (The sum is undefined if **A** and **B** have different sizes.) We can also define the multiplication of a matrix by a scalar: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$. Matrix multiplication (the product of two matrices) is more complicated. The product **AB** is defined only if **A** is of size $a \times b$ and **B** is of size $b \times c$ (i.e., the second matrix has the same number of rows as the first has columns); the result is a matrix of size $a \times c$. If the matrices are of appropriate size, then the result is

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j}\mathbf{B}_{j,k} \, .$$

Matrix multiplication is not commutative, even for square matrices: $\mathbf{AB} \neq \mathbf{BA}$ in general. It is, however, associative: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Note that the dot product can be expressed in terms of a transpose and a matrix multiplication:$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\top \mathbf{y}$.

IDENTITY MATRIX
TRANSPOSE
INVERSE
SINGULAR

The **identity matrix I** has elements $\mathbf{I}_{i,j}$ equal to 1 when $i = j$ and equal to 0 otherwise. It has the property that $\mathbf{AI} = \mathbf{A}$ for all **A**. The **transpose** of **A**, written $\mathbf{A}^\top$ is formed by turning rows into columns and vice versa, or, more formally, by $\mathbf{A}^\top{}_{i,j} = \mathbf{A}_{j,i}$. The **inverse** of a square matrix **A** is another square matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. For a **singular** matrix, the inverse does not exist. For a nonsingular matrix, it can be computed in $O(n^3)$ time.

Matrices are used to solve systems of linear equations in $O(n^3)$ time; the time is dominated by inverting a matrix of coefficients. Consider the following set of equations, for which we want a solution in x, y, and z:

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \, . \end{aligned}$$

We can represent this system as the matrix equation $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, where

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}.$$

To solve $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ we multiply both sides by $\mathbf{A}^{-1}$, yielding $\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, which simplifies to $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. After inverting $\mathbf{A}$ and multiplying by $\mathbf{b}$, we get the answer

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

A.3 Probability Distributions

A probability is a measure over a set of events that satisfies three axioms:

1. The measure of each event is between 0 and 1. We write this as $0 \leq P(X = x_i) \leq 1$, where X is a random variable representing an event and x_i are the possible values of X. In general, random variables are denoted by uppercase letters and their values by lowercase letters.
2. The measure of the whole set is 1; that is, $\sum_{i=1}^{n} P(X = x_i) = 1$.
3. The probability of a union of disjoint events is the sum of the probabilities of the individual events; that is, $P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$, where x_1 and x_2 are disjoint.

A **probabilistic model** consists of a sample space of mutually exclusive possible outcomes, together with a probability measure for each outcome. For example, in a model of the weather tomorrow, the outcomes might be *sunny, cloudy, rainy*, and *snowy*. A subset of these outcomes constitutes an event. For example, the event of precipitation is the subset consisting of $\{$*rainy, snowy*$\}$.

We use $\mathbf{P}(X)$ to denote the vector of values $\langle P(X = x_1), \ldots, P(X = x_n)\rangle$. We also use $P(x_i)$ as an abbreviation for $P(X = x_i)$ and $\sum_x P(x)$ for $\sum_{i=1}^{n} P(X = x_i)$.

The conditional probability $P(B|A)$ is defined as $P(B \cap A)/P(A)$. A and B are conditionally independent if $P(B|A) = P(B)$ (or equivalently, $P(A|B) = P(A)$). For continuous variables, there are an infinite number of values, and unless there are point spikes, the probability of any one value is 0. Therefore, we define a **probability density function**, which we also denote as $P(\cdot)$, but which has a slightly different meaning from the discrete probability function. The density function $P(x)$ for a random variable X, which might be thought of as $P(X = x)$, is intuitively defined as the ratio of the probability that X falls into an interval around x, divided by the width of the interval, as the interval width goes to zero:

PROBABILITY DENSITY FUNCTION

$$P(x) = \lim_{dx \to 0} P(x \leq X \leq x + dx)/dx\,.$$