FROM: by LESLIE VALIANT

starting point n eventually reaches the value $n=1$ and terminates. Many starting points have been tried since the mathematician Lothar Collatz posed the problem in 1937. They all resulted in computations that did terminate at $n=1$. But—somewhat shockingly, given how simple the problem is to describe—no one has been able to offer a proof that this process would terminate for every possible starting point, or that it would not.

Collatz's problem is an example of apparent inherent complexity in simple procedures, even those isolated from any complex environment. In this case the notion of input can be removed altogether by considering a compound procedure that feeds the starting numbers $n=2$, 3, 4 in succession to the basic procedure, going on to the next starting number when the sequence generated by the previous one has terminated at $n=1$. Asking whether this compound procedure will ever get to every starting number n, rather than get stuck in perpetuity after a specific n, is equivalent to the original problem. In this light we should not be so shocked by the noncomputability of the Halting Problem, which would need to be able to make some kind of prediction about the ultimate fate not just of one, but of any computation.

3.7 The Perceptron Algorithm

Our journey through the major themes of computational complexity now brings us finally to the vicinity of our destination, the study of ecorithms. Our final point of departure is a simple but important algorithm that, like Collatz's problem, can also have complicated behavior, but these complications can be attributed to the outside environment in which it operates. This example is the perceptron algorithm, proposed by Frank Rosenblatt in the 1950s.[18]

The perceptron algorithm operates in the following context. Assume that there is a set of potential examples, each one specified by some description, and further that there is a criterion for which some of the examples are true examples and the others false. For instance, an example may be an individual flower, and the criterion may be whether that flower belongs to species A or species B.[19] The perceptron algorithm requires that the examples be described somehow. For this case, let us say that the description consists of two numbers x and y that specify the length and width of one of its petals.

The perceptron algorithm is a member of the class of supervised learning algorithms, which means it can be trained to do the work of classifying ex-

BUY THE BOOK

amples according to our criterion and descriptions. First, the algorithm is given descriptions of a set of training examples, as well as the correct label of each. For instance, one flower may have a petal 3 units long and 1 unit wide, and be labeled as a member of species *A*. In a subsequent phase the algorithm is fed with a set of test examples, which consist of descriptions of examples but no labels. The goal of the algorithm is to predict reliably for each test example whether it is true or false, or as in the flower case, an instance of species *A* or species *B*.

The perceptron algorithm works when there is a certain mathematical criterion, known as a linear separator, dividing the two possible classes. This criterion, in the case of our flowers, is a rule of the form

$$px+qy>r$$

where p, q, and r are numbers, such that every flower that satisfies it is of species *A*, and every one that does not is of species *B*. For example, suppose that $p=2$, $q=-3$ and $r=2$, so that the rule is

$$2x-3y>2.$$

Then a flower with petal length 5 and width 2 would be classified as type *A* since $(2\times5)-(3\times2)=4$, and $4>2$. On the other hand a flower with petal length 3 and width 2 would be classified as type *B* since $(2\times3)-(3\times2)=0$ and $0<2$. In graphical terms this means that if all the examples are plotted in two dimensions, representing the length by x and the width by y, then there is a straight line corresponding to equation $2x-3y=2$, so that all the species *A* flowers lie on one side of this line, and the species *B* flowers on the other (or on it). This is illustrated in Figure 3.6.

Of course, the perceptron algorithm does not know the true equation for the separator in advance. Instead, it must find it. The algorithm works by scanning through the training data, possibly many times. At each instant it maintains a hypothesis, of the form of $ax+by>c$, about the linear separator. We shall for simplicity work with the case $c=0$.[20] The algorithm then starts with the hypothesis $0x+0y>0$. It goes through each training example one by one, and if the example label is correctly predicted by the current hypothesis, then the hypothesis is not changed. If the example label is not predicted correctly, then the hypothesis is updated so as to be "more likely,"

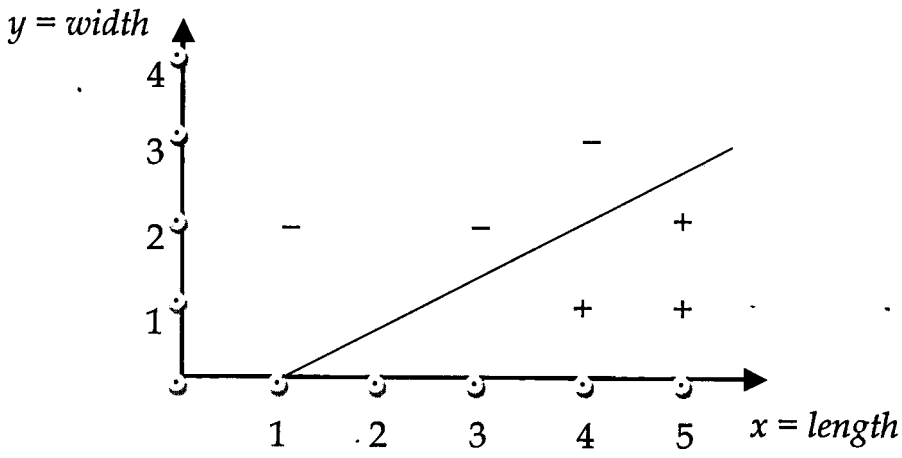

Figure 3.6 The sloping line contains the points satisfying $2x-3y=2$. The points $(x=4, y=1)$, $(x=5, y=1)$, and $(x=5, y=2)$ all satisfy $2x-3y>2$ and are marked as "+," while the points $(x=1, y=2)$, $(x=3, y=2)$, and $(x=4, y=3)$ do not and are marked as "–." In other words, the flowers of species A will lie below the line, and those of species B above or on it.

in a certain sense, to be correct on that same example if presented again later.

To be more precise, if the hypothesis misclassifies a true positive example (u, v) as negative (i.e., because $au+bv\leq 0$), then a is updated by having u added to it, and b by having v added to it. The left-hand side of the updated hypothesis will then be $(a+u)x+(b+v)y$, and it will have value $(a+u)u+(b+v)v$ if presented with the same example (u, v) on a subsequent run through the data. The value of the sum will be larger than before by a positive quantity u^2+v^2, and hence will be "more likely" to exceed 0 in value and correctly identify the positive example as true. For the opposite case, when a negative example is misclassified to be positive, a is updated by having u subtracted from it, and b by having v subtracted from it. This will have the effect of reducing the value of the left-hand side by u^2+v^2 if the same input (u, v) is presented again later, and hence will make it "more likely" to be less than 0 and hence result in a correct negative classification in that eventuality.

Depending on the order in which training data is fed to the perceptron, it could generate exceedingly many distinct histories of hypotheses. The interesting fact about the perceptron algorithm is that, in spite of our lack of control over its exact fate as we let it loose on arbitrary data, it nonetheless

manages to achieve something quite remarkable. The most basic statement of the power of this algorithm, proved by Albert Novikoff soon after the algorithm was first proposed, is that if there is a true linear separator, then the algorithm is sure to find it, or another hypothesis that also correctly classifies all the examples, after having made misclassifications only a finite number of times. Furthermore, an upper bound on the number of such misclassifications can be computed given the data. This upper bound is equal to M/m^2, where M is the square of the distance of the furthest data point in the training set from the point (0, 0) and m is the margin. The margin has a trickier definition: It is the minimum distance of any data point from the separating line for the line for which this distance is the largest. The consequence of m being in the denominator is that the closer the data points are to the separator, the more mistakes this procedure can potentially make.

The importance of the algorithm derives from several additional facts. First, it works within bounds of the form M/m^2 not just for problems with two variables but for problems with any number of variables. Second, in practice, it often works well even for data that is corrupted by noise. Third, there are general methods for dealing with data that is separable not by linear relations but by more complex curves. For example, suppose that the two categories are not separated by a straight line as they are in Figure 3.6, but we suspect that some more complex curve would separate them. In our two-dimensional case we could try to learn the separator $ax + by + cxy + dx^2 + ey^2 > f$ where x, y are variables and a, b, c, d, and e are the constants to be learned. This inequality is not linear in x, y, since it contains higher order terms such as x^2. However, it can be viewed as linear if we regard the set of variables not as $\{x, y\}$ but as $\{x, y, xy, x^2, y^2\}$. We can translate any example given as a pair $\{x, y\}$ of numbers to the corresponding five numbers $\{x, y, xy, x^2, y^2\}$ by multiplication. In this way the perceptron algorithm can be applied directly to nonlinearly separable data also.

This linearization is an important idea that greatly extends the range of applicability of the perceptron algorithm, but it is not the complete panacea that it may seem. If there are few nonlinear terms, and we know which they are, then there are no problems. But if there are numerous terms potentially to look for, then this will introduce higher, possibly exponential, costs.

The criterion that only a finite number of mistakes are made over any, even infinite, number of examples does not appear to be a natural fit for human learning. It raises the question of what outcome we should really require

True Value	Example	Classification by Previous Hypothesis	Updated Hypothesis
			$0x+0y+0z>0$
+	(4, 1, 1)	−	$4x+1y+1z>0$
−	(1, 2, 1)	+	$3x-1y+0z>0$
+	(5, 1, 1)	+	$3x-1y+0z>0$
−	(3, 2, 1)	+	$0x-3y-1z>0$
+	(5, 2, 1)	−	$5x-1y+0z>0$
−	(4, 3, 1)	+	$1x-4y-1z>0$
+	(4, 1, 1)	−	$5x-3y+0z>0$
−	(1, 2, 1)	−	$5x-3y+0z>0$
+	(5, 1, 1)	+	$5x-3y+0z>0$
−	(3, 2, 1)	+	$2x-5y-1z>0$
+	(5, 2, 1)	−	$7x-3y+0z>0$
−	(4, 3, 1)	+	$3x-6y-1z>0$
+	(4, 1, 1)	+	$3x-6y-1z>0$
−	(1, 2, 1)	−	$3x-6y-1z>0$
+	(5, 1, 1)	+	$3x-6y-1z>0$
−	(3, 2, 1)	−	$3x-6y-1z>0$
+	(5, 2, 1)	+	$3x-6y-1z>0$
−	(4, 3, 1)	−	$3x-6y-1z>0$

Figure 3.7 Example of a run of the perceptron algorithm in three dimensions on the set of six examples +(4, 1, 1), −(1, 2, 1), +(5, 1, 1), −(3, 2, 1), +(5, 2, 1), −(4, 3, 1) repeated in that order three times. The signs indicate the labels of the examples. The initial hypothesis is $0x+0y+0z>0$. The first example (4, 1, 1) when substituted in the left-hand side of the initial hypothesis gives 0, and hence does not satisfy it, as indicated by the negative sign in the third column. The first column indicates that the true label of this first example (4, 1, 1) is positive. The algorithm therefore adds the coordinates (4, 1, 1) of the example to the coefficients (0, 0, 0) of the hypothesis, to give $4x+1y+1z>0$ as the updated hypothesis. After the six examples are cycled through twice, the hypothesis $3x-6y-1z>0$ is obtained. In the third cycle it is confirmed that this hypothesis satisfies all six examples.

of a learning algorithm before we declare it successful. This is the main question that will be addressed in Chapter 5. Before we get there, however, we need to take a more general look at what a computationally sound, mechanistic explanation of a natural phenomenon—whether of evolution, or cognition, or some other process of interest—might look like.

But there is one intuition suggested by the perceptron algorithm that will be important for what comes later. Learning is achieved in many steps that are plausible but innocuous when viewed one by one in isolation. These steps work because there is an overall algorithmic plan. In combination the steps achieve something, in particular, some kind of convergence. We shall claim that evolution is similar. The many small steps taken do not make too much sense one by one. But there is an algorithmic plan, so that taken in unison the many steps do achieve something remarkable.