Computational Learning Theory for Artificial Neural Networks

Martin Anthony and Norman Biggs

Department of Statistical and Mathematical Sciences, London School of Economics and Political Science, Houghton St., London WC2A 2AE, United Kingdom

1. LEARNING AND TRAINING: A FRAMEWORK

There are many types of activity which are commonly known as 'learning'. Here, we shall discuss a mathematical model of one such process, known as the the 'probably approximately correct' (or PAC) model. We shall illustrate how key problems of learning in artificial neural networks can be studied within this framework, presenting theoretical analyses of two important issues: the size of training sample that should be used, and the running time of learning algorithms; in other words, *sample complexity* and *computational complexity.*

In a general framework for our discussion of learning, we have a 'real world' W containing a set of objects which we shall refer to as examples. We also have a 'pre-processor' P, which takes an example and converts it into a coded message, such as a string of bits or a real vector. This coded version of the example is then presented to M, a machine whose purpose is to recognise certain examples. The output of M is a single bit, either 1 (if the example is recognised as belonging to a certain set), or 0 (if not). For example, M may be a feedforward linear threshold network. The machine can be in one of many states. For example, M might be in a state in which it will recognise examples of the letter **A**, coded as a string of bits in which the 1's correspond to the positions of black pixels on a grid; other states of M might enable it to recognise examples of other letters. The learning process we consider involves making changes to the state of M on the basis of examples presented to it, so that it achieves some desired classification. For instance, we might be able to change the state of M in such a way that the more positive and negative examples of the letter **A** that are presented, the better will M recognise future examples of that letter.

For our purposes, a 'concept' can be described by a set of examples. Let Σ be a set, which we shall call the *alphabet* for describing examples. In this article, Σ will be either the *boolean* alphabet $\{0, 1\}$, or the *real* alphabet $\mathbf{R}$. We denote the set of n-tuples of elements of Σ by Σ^n and the set of all non-empty finite strings of elements of Σ by Σ^*. Let X be a subset of Σ^*. We define a *concept*, over the alphabet Σ, to be a function $c : X \to \{0, 1\}$. In some cases X will be the whole of Σ^*; while in other cases X will be taken as Σ^n for one specific value of n, which will be clear from the context.

The set X will be referred to as the *example space*, and its members as *examples.* An example $y \in X$ for which $c(y) = 1$ is known as a *positive example*, and an example for which $c(y) = 0$ is known as a *negative example.* The union of the set of positive examples and the set of negative examples is the domain of c. So, provided that the domain is known, c determines, and is determined by, its set of positive examples. Sometimes it is helpful to think of a concept as a set in that way.

There are two sets of concepts inherent in our framework for learning. First, there is the set of concepts derived from the real world which it is proposed to recognise. This set might contain concepts like 'the letter **A**', 'the letter **B**', 'the letter **C**', and so on, each of which can be coded to determine a set of positive and negative examples. When a set of concepts is determined in this way, we shall use the term *concept space* for it. Secondly, there is the set of concepts which the machine M is capable of recognising. We shall suppose that M can assume various states, and that in a given state it will classify some inputs as positive examples (output 1), and the rest as negative examples (output 0). Thus a state of M determines a concept, which we may think of as a hypothesis. For this reason, the set of all concepts which M determines will be referred to as its *hypothesis space.*

Generally speaking, we are given two sets of concepts, C (the concept space) and H (the hypothesis space), and the problem is to find, for each $c \in C$, some $h \in H$ which is a good approximation to c. In realistic situations hypotheses are formed on the basis of certain information which does not amount to an explicit definition of c. In the framework we describe, we shall assume that this information is provided by a sequence of positive and negative examples of c. In practice there are constraints upon our resources, and we have to be content with a hypothesis h which 'probably' represents c 'approximately', in some sense to be defined.

Let $X \subseteq \Sigma^*$ be the example space, where as always Σ is $\{0, 1\}$ or $\mathbf{R}$. A *sample* of *length* m is just a sequence of m examples, that is, an m-tuple $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ in X^m. A *training sample* $\mathbf{s}$ is an element of $(X \times \{0, 1\})^m$, that is,

$$\mathbf{s} = ((x_1, b_1),\ (x_2, b_2),\ \ldots,\ (x_m, b_m)),$$

where the x_i are examples and the b_i are bits. We say that $\mathbf{s}$ is a *training sample for the target concept* t if $b_i = t(x_i),\ (1 \le i \le m)$.

A *learning algorithm for* (C, H) (or a (C, H)*-learning algorithm*) is a procedure which accepts training samples for functions in C and outputs corresponding hypotheses in

H. Of course, in order to qualify as an algorithm the procedure must be effective in some sense; we shall discuss this point in more detail in due course. If we ignore the problem of effectiveness, a learning algorithm is simply a function L which assigns to any training sample $\mathbf{s}$ for a target concept $t \in C$ a function $h \in H$. We write $h = L(\mathbf{s})$. In what follows, we generally assume that $C \subseteq H$.

If there are no contradictory labels in the sample, then $\mathbf{s}$ may be regarded as a function defined on the set $\{x_1, x_2, \ldots, x_m\}$, given by $\mathbf{s}(x_i) = b_i$ $(1 \leq i \leq m)$. The hypothesis $L(\mathbf{s})$ is a function defined on the whole example space X. A hypothesis h in H is said to be *consistent* with $\mathbf{s}$, or to *agree* with $\mathbf{s}$, if $h(x_i) = b_i$ for each $1 \leq i \leq m$. We do not, in general, make the assumption that $L(\mathbf{s})$ is consistent with $\mathbf{s}$, but when this condition does hold for all $\mathbf{s}$ we shall say that L itself is *consistent.* In that case the function $L(\mathbf{s})$ is simply an extension of $\mathbf{s}$.

As indicated above, the machines needed in this area of Computational Learning Theory have the additional feature that they can take on a number of different states, and so a machine represents a set of functions, rather than a single function. A state of the machine is a representation of a function, and the set of all such functions comprises the hypothesis space defined by the machine. For example, in the case in which M is an artificial neural network, the states of the machine are the possible weight assignments. It is convenient to define a *representation* to be a surjection $\Omega \to H$, where Ω is a set and H is a hypothesis space. The set Ω may be the set of states of a machine. The surjection assigns to each state ω a corresponding function h_ω.

Example Consider the *linear threshold machine*, or *boolean perceptron*, with n boolean inputs $y_1, y_2, \ldots, y_n \in \{0, 1\}$ and a single active node. The arcs carrying the inputs have associated weights $\alpha_1, \ldots, \alpha_n$ and the weighted sum $\sum_{i=1}^{n} \alpha_i y_i$ of the inputs is applied to the active node. This node outputs 1 if the weighted sum is at least θ, and 0 otherwise. In this case the representation $\Omega \to H$ is defined as follows. The set Ω is the real space $\mathbf{R}^{n+1}$, that is, the set of $(n+1)$-tuples $\omega = (\alpha_1, \ldots, \alpha_n, \theta)$. The function h_ω is given by $h_\omega(y_1 y_2 \ldots y_n) = 1$ if and only if $\sum_{i=1}^{n} \alpha_i y_i \geq \theta$. $\square$

2. THE BASIC PAC MODEL

In general, not every extension of a training sample will be a valid generalisation, because the target concept is only partially defined by the sample. Furthermore, a training sample may be unrepresentative or misleading. This prompted Valiant (1984a, 1984b) to propose a probabilistic model of learning, which we now describe. We start by illustrating the basic ideas by means of a very simple example.

For each real number θ the *ray* r_θ is the concept defined on $\mathbf{R}$ by $r_\theta(y) = 1$ if and only if $y \geq \theta$. An algorithm for learning in the hypothesis space $H = \{r_\theta \mid \theta \in \mathbf{R}\}$ is based on the idea of taking the current hypothesis to be the 'smallest' ray containing all the positive examples in the training sample. A suitable default hypothesis when there are no positive examples in the sample will be the identically-0 function. For convenience, we therefore consider this to be a ray, and call it the *empty ray*. It will be denoted r_∞, where ∞ is merely a symbol taken to be greater than any real number. Then, for a given training sample $\mathbf{s} = ((x_1, b_1), \ldots, (x_m, b_m))$, the output hypothesis $L(\mathbf{s})$ should be r_λ, where

$$\lambda = \lambda(\mathbf{s}) = \min_{1 \le i \le m} \{x_i \mid b_i = 1\},$$

and $\lambda = \infty$ if the sample contains no positive examples.

It is easy to see that if the training sample is for a target concept r_θ then $L(\mathbf{s})$ will be a ray r_λ with $\lambda = \lambda(\mathbf{s}) \ge \theta$. Because there are only finitely many examples in a training sample, and the example space is uncountable, we cannot expect that $\lambda = \theta$. However, it seems that as the length of the training sample increases, so should the likelihood that there is small error resulting from using r_λ instead of r_θ.

In practical terms, this property can be characterised as follows. Suppose we run the algorithm with a large training sample, and then decide to use the output hypothesis r_λ as a substitute for the (unknown) target concept r_θ. In other words, we are satisfied that the 'learner' has been adequately trained. If λ is not close to θ, this indicates that positive examples close to θ are relatively unlikely and did not occur in the training sample. Consequently, if we now classify some more examples which are presented according to the same distribution, then we shall make few mistakes as a result of using r_λ instead of r_θ.

Consider a general learning framework in which a training sample $\mathbf{s}$ for a target concept t is generated by drawing the examples $x_1, x_2, \ldots, x_m$ from X 'at random', according to some fixed, but unknown, probability distribution. A learning algorithm L produces a hypothesis $L(\mathbf{s})$ which, it is hoped, is a good approximation to t. More fully, we require that, as the number m of examples in the training sample increases, so does the likelihood that the error which results from using $L(\mathbf{s})$ in place of t is small.

To formalise this, we suppose that we have a probability space (X, Σ, μ). Here, Σ is a σ-algebra of subsets of X and μ is a probability measure. In the cases under discussion here, the example space X is boolean or real. In the boolean case, we take Σ to be the set of all subsets of X and in the real case, $X \subseteq \mathbf{R}^n$, we take Σ to be the Borel algebra on X. (Those unfamiliar with the notions just discussed may find the details in Billingsley (1986), for example.) In both cases, we use the appropriate σ-algebra without explicit reference to the details. From now on, then, we shall simply speak of a 'probability distribution μ on X', by which we mean a function μ defined on the appropriate family Σ and satisfying the axioms given above. It must be emphasised that, in the applications we have in mind, we make no assumptions about μ, beyond the conditions stated in the definition. The situation we are modelling is that of a world of examples presented to the learner according to some fixed but unknown distribution. The 'teacher' is allowed to classify the examples as positive or negative, but cannot control the sequence in which the examples are presented.

We shall continue to assume that $C \subseteq H$, so that the target concept belongs to a hypothesis space H which is available to the learner. Given a target concept $t \in H$ we define the *error* of any hypothesis h in H, with respect to t, to be the probability of the event $h(x) \neq t(x)$. That is,

$$\mathrm{er}_\mu(h,t) = \mu\{x \in X \mid h(x) \neq t(x)\}.$$

We refer to the set on the right-hand side as the *error set*, and we assume that it is an event, so that a probability can be assigned to it. This can be guaranteed when the hypothesis space is *universally separable*; see Blumer *et al.* (1989). In order to streamline the notation, we suppress the explicit reference to t when it is clear from the context, and we write $\mathrm{er}_\mu(h)$ in place of $\mathrm{er}_\mu(h,t)$.

When a given set X is provided with the structure of a probability space, the product set X^m inherits a probability space structure from X. The corresponding probability distribution on X^m is the *product distribution* μ^m. Informally, for a given $Y \subseteq X^m$ we shall interpret the value $\mu^m(Y)$ as 'the probability that a random sample of m examples drawn from X according to the distribution μ belongs to Y'.

Let $S(m,t)$ denote the set of training samples of length m for a given target concept t, where the examples are drawn from an example space X. Any sample $\mathbf{x} \in X^m$ determines, and is determined by, a training sample $\mathbf{s} \in S(m,t)$: if $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, then $\mathbf{s} = ((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m)))$. In other words, there is a bijection $\phi : X^m \to S(m,t)$ for which $\phi(\mathbf{x}) = \mathbf{s}$. Thus, we can interpret the probability that $\mathbf{s} \in S(m,t)$ has some given property P in the following way. We define

$$\mu^m\{\mathbf{s} \in S(m,t) \mid \mathbf{s} \text{ has property } P\} = \mu^m\{\mathbf{x} \in X^m \mid \phi(\mathbf{x}) \in S(m,t) \text{ has property } P\}.$$

It follows that, when the example space X is equipped with a probability distribution μ, we can give a precise interpretation to:

(i) the error of the hypothesis produced when a learning algorithm L is given $\mathbf{s}$;
(ii) the probability that this error is less than ϵ.

The first quantity is just $\mathrm{er}_\mu(L(\mathbf{s}))$. The second is the probability, with respect to μ^m, that $\mathbf{s}$ has the property $\mathrm{er}_\mu(L(\mathbf{s})) < \epsilon$. Putting all this together we can formulate the notion that, given a *confidence* parameter δ and an *accuracy* parameter ϵ, the probability that the error is less than ϵ is greater than $1-\delta$. The result is one of the key definitions in this field. It was formulated first by Valiant (1984a) and, using this terminology, by Angluin (1988).

We say that the algorithm L is a *probably approximately correct* learning algorithm for (C,H) if, given

- a real number δ $(0 < \delta < 1)$;
- a real number ϵ $(0 < \epsilon < 1)$;

then there is a positive integer $m_0 = m_0(\delta, \epsilon)$ such that

- for any target concept $t \in C$, and
- for any probability distribution μ on X,

whenever $m \geq m_0$, $\mu^m\{\mathbf{s} \in S(m,t) \mid \text{er}_\mu(L(\mathbf{s})) < \epsilon\} > 1 - \delta$.

The term 'probably approximately correct' is usually abbreviated to the acronym *pac*.

The fact that m_0 depends upon δ and ϵ, but not on t and μ, reflects the fact that the learner may be able to specify the desired levels of confidence and accuracy, even though the target concept and the distribution of examples are unknown. The reason that it is at all possible to satisfy the condition for any μ is that it expresses a relationship between two quantities which involve μ: the error er_μ and the probability with respect to μ^m of a certain set.

Pac learning is, in a sense, the best one can hope for within this probabilistic framework. Unrepresentative training samples, although unlikely, will on occasion be presented to the learning algorithm, and so one can only expect that it is *probable* that a useful training sample is presented. In addition, even for a representative training sample, an extension of the training sample will not generally coincide with the target concept, so one can only expect that the output hypothesis is *approximately* correct.

For illustration, we now give a formal verification that the algorithm described above is a pac learning algorithm for rays.

Theorem 1 *The learning algorithm L for rays given above is probably approximately correct.*

Proof Suppose that $\delta, \epsilon, r_\theta$, and μ are given. Let $\mathbf{s}$ be a training sample of length m for r_θ and let $L(\mathbf{s}) = r_\lambda$. Clearly, the error set is the interval $[\theta, \lambda)$. For the given value of ϵ, and the given μ, define $\beta_0 = \beta_0(\epsilon, \mu) = \sup\{\beta \mid \mu[\theta, \beta) < \epsilon\}$. Then it follows that $\mu[\theta, \beta_0) \leq \epsilon$ and $\mu[\theta, \beta_0] \geq \epsilon$. Thus if $\lambda \leq \beta_0$ we have

$$\text{er}_\mu(L(\mathbf{s})) = \mu[\theta, \lambda) \leq \mu[\theta, \beta_0) \leq \epsilon.$$

The event that $\mathbf{s}$ has the property $\lambda \leq \beta_0$ is just the event that at least one of the examples in $\mathbf{s}$ is in the interval $[\theta, \beta_0]$. Since $\mu[\theta, \beta_0] \geq \epsilon$ the probability that a single example is not in this interval is at most $1 - \epsilon$. Therefore the probability that none of the m examples comprising $\mathbf{s}$ is in this interval is at most $(1 - \epsilon)^m$. Taking the complementary event, it follows that the probability that $\lambda \leq \beta_0$ is at least $1 - (1 - \epsilon)^m$. We noted above that the event $\lambda \leq \beta_0$ implies the event $\text{er}_\mu(L(\mathbf{s})) \leq \epsilon$, and so

$$\mu^m\{\mathbf{s} \in S(m, r_\theta) \mid \text{er}_\mu(L(\mathbf{s})) \leq \epsilon\} \geq 1 - (1 - \epsilon)^m.$$

Note that the right-hand side is independent of the target function r_θ (and μ). In order to make it greater than $1 - \delta$ we can take

$$m \geq m_0 = \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\delta} \right\rceil.$$

For then it follows that

$$(1-\epsilon)^m \leq (1-\epsilon)^{m_0} < \exp(-\epsilon m_0) < \exp(\ln \delta) = \delta.$$

This calculation shows that the algorithm is pac. (The inequality $\text{er}_\mu(L(\mathbf{s})) \leq \epsilon$ is obtained, whereas our definition of pac learning requires the inequality to be strict. However, it is easy to see that this makes no difference.) □

This provides an explicit formula for the length of sample (that is, the amount of training) sufficient to ensure prescribed levels of confidence and accuracy. Suppose we require $\delta = 0.001$ and $\epsilon = 0.01$. Then the value of m_0 is $\lceil 100 \ln 1000 \rceil = 691$. So if we supply at least 691 labelled examples of any ray (these examples being chosen at random according to a fixed distribution μ) and take the output ray as a substitute for the target, then we can be 99.9% sure that at most 1% of future examples will be classified wrongly, provided they are drawn from the same source as the training sample.

It is convenient at this stage to define the notion of 'observed error'. For a given training sample

$$\mathbf{s} = ((x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m)),$$

we define the *observed error* of h on $\mathbf{s}$ to be

$$\text{er}_\mathbf{s}(h) = \frac{1}{m} |\{i \mid h(x_i) \neq b_i\}|.$$

When $\mathbf{s}$ is a training sample for $t \in C$, the observed error of h on $\mathbf{s}$ is the fraction of examples in the sample which h and t classify differently. Recall that a learning algorithm L for (C, H) is consistent if, given any training sample $\mathbf{s}$ for a target concept $t \in H$, the output hypothesis $h = L(\mathbf{s}) \in H$ agrees with t on the examples in $\mathbf{s}$. In terms of observed error, a learning algorithm is consistent if for all $t \in C$ and all m, given any $\mathbf{s} \in S(m, t)$, we have $\text{er}_\mathbf{s}(L(\mathbf{s})) = 0$.

As usual, assume that there is an unknown probability distribution μ on X. Suppose we fix, for the moment, a target concept $t \in C$. Given $\epsilon \in (0, 1)$ the set of $h \in H$ for which $\text{er}_\mu(h) \geq \epsilon$ may be described as the set of *ϵ-bad* hypotheses for t. A consistent algorithm for (C, H) produces an output hypothesis h such that $\text{er}_\mathbf{s}(h) = 0$, and the pac property requires that such an output is unlikely to be ϵ-bad. This leads to the following definition.

We say that the hypothesis space H is *potentially learnable* if, given real numbers δ and ϵ $(0 < \delta, \epsilon < 1)$, there is a positive integer $m_0 = m_0(\delta, \epsilon)$ such that, whenever $m \geq m_0$,

$$\mu^m\{\mathbf{s} \in S(m, t) \mid \text{for all } h \in H,\ \text{er}_\mathbf{s}(h) = 0 \Longrightarrow \text{er}_\mu(h) < \epsilon\} \ > \ 1 - \delta,$$

for any probability distribution μ on X and any $t \in H$. The following result is clear.

Theorem 2 *If H is potentially learnable, and L is a consistent learning algorithm for H, then L is pac.*

The definition of potential learnability is quite complex, but we have the following standard result, observed by Blumer *et al.* (1989), for example.

Theorem 3 *Any finite hypothesis space is potentially learnable.*

Proof If h has error at least ϵ then, $\mu\{x \in X \mid h(x) = t(x)\} = 1 - \mathrm{er}_\mu(h) \leq 1 - \epsilon$. Thus

$$\mu^m\{\mathbf{s} \mid h(x_i) = t(x_i)\ (1 \leq i \leq m)\} \leq (1-\epsilon)^m.$$

This is the probability that any one ϵ-bad hypothesis is consistent with $\mathbf{s}$. The probability that there is *some* ϵ-bad consistent hypothesis is therefore less than $|H|(1-\epsilon)^m$. This is less than δ if

$$m \geq m_0 = \left\lceil \frac{1}{\epsilon} \ln \frac{|H|}{\delta} \right\rceil.$$

Taking the complementary event we have

$$\mu^m\{\mathbf{s} \in S(m,t) \mid \text{for all } h \in H,\ \mathrm{er}_\mathbf{s}(h) = 0 \Longrightarrow \mathrm{er}_\mu(h) < \epsilon\} \ > \ 1 - \delta,$$

for $m \geq m_0$, which is the required conclusion. □

It is clear that this is a useful theorem. It covers all boolean cases, where the example space is $\{0,1\}^n$ (or a subset thereof) with n fixed. In any such situation a consistent algorithm is automatically pac.

3. THE VAPNIK-CHERVONENKIS DIMENSION

Suppose that we have a hypothesis space H defined on an example space X. We have seen that if H is finite, then it is potentially learnable. The proof depends critically on the finiteness of H and cannot be extended to provide results for infinite H. However, there are many situations where the hypothesis space is infinite, and it is desirable to extend the theory to cover this case. A pertinent comment is that most hypothesis spaces which occur 'naturally' have a high degree of structure, and even if the space is infinite it may contain functions only of a special type. This is true, almost by definition, for any hypothesis space H which is constructed by means of a representation $\Omega \to H$.

The key to extending results on potential learnability to infinite spaces is the observation that what matters is not the cardinality of H, but rather what may be described as its 'expressive power'. This notion can be formalised by means of the *Vapnik-Chervonenkis dimension* of H, a notion originally defined by Vapnik and Chervonenkis (1971), and introduced into learnability theory by Blumer *et al.* (1989).

In order to illustrate some of the ideas, we consider the *real perceptron.* This is a machine which operates in the same manner as the linear threshold machine, but with real-valued inputs. Thus, there are n inputs and a single active node. The arcs carrying the inputs have real-valued weights $\alpha_1, \alpha_2, \ldots, \alpha_n$ and there is a real threshold value θ at the active node. As with the linear threshold machine, the weighted sum of the inputs is applied to the active node and this node outputs 1 if and only if the weighted sum is at least the threshold value θ. More precisely, the real perceptron P_n on n inputs is defined by means of a representation $\Omega \to H$, where the set of states Ω is $\mathbf{R}^{n+1}$. For a state $\omega = (\alpha_1, \alpha_2, \ldots, \alpha_n, \theta)$, the function $h_\omega \in H$, from $X = \mathbf{R}^n$ to $\{0,1\}$, is given by $h_\omega(y) = 1$ if and only if $\sum_{i=1}^{n} \alpha_i y_i \geq \theta$. It should be noted that $\omega \mapsto h_\omega$ is not an injection: for any $\lambda > 0$ the state $\lambda\omega$ defines the same function as ω.

Suppose that H is a hypothesis space defined on the example space X, and let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ be a sample of length m of examples from X. We define $\Pi_H(\mathbf{x})$, the *number of classifications of* $\mathbf{x}$ *by* H, to be the number of distinct vectors of the form

$$(h(x_1), h(x_2), \ldots, h(x_m)),$$

as h runs through all hypotheses of H. Although H may be infinite, $H|_{\mathbf{x}}$, the hypothesis space obtained by restricting the hypotheses of H to domain $E_{\mathbf{x}} = \{x_1, x_2, \ldots, x_m\}$, is finite and is of cardinality $\Pi_H(\mathbf{x})$. Note that for any sample $\mathbf{x}$ of length m, $\Pi_H(\mathbf{x}) \leq 2^m$. An important quantity, and one which shall turn out to be crucial in applications to potential learnability, is the maximum possible number of classifications by H of a sample of a given length. We define the *growth function* Π_H by

$$\Pi_H(m) = \max\{\Pi_H(\mathbf{x}) : \mathbf{x} \in X^m\}.$$

We have used the notation Π_H for both the number of classifications and the growth function, but this should cause no confusion.

We noted above that the number of possible classifications by H of a sample of length m is at most 2^m, this being the number of binary vectors of length m. We say that a sample $\mathbf{x}$ of length m is *shattered* by H, or that H *shatters* $\mathbf{x}$, if this maximum possible value is attained; that is, if H gives all possible classifications of $\mathbf{x}$. Note that if the examples in $\mathbf{x}$ are not distinct then $\mathbf{x}$ cannot be shattered by any H. When the examples are distinct, $\mathbf{x}$ is shattered by H if and only if for any subset S of $E_{\mathbf{x}}$, there is some hypothesis h in H such that for $1 \leq i \leq m$, $h(x_i) = 1 \Longleftrightarrow x_i \in S$. S is then the subset of $E_{\mathbf{x}}$ comprising the positive examples of h.

Based on the intuitive notion that a hypothesis space H has high expressive power if it can achieve all possible classifications of a large set of examples, we use as a measure of this power the *Vapnik-Chervonenkis dimension*, or *VC dimension*, of H, defined as follows. The VC dimension of H is the maximum length of a sample shattered by H; if there is no such maximum, we say that the VC dimension of H is infinite. Using the notation introduced in the previous section, we can say that the VC dimension of H, denoted $\mathrm{VCdim}(H)$, is given by

$$\mathrm{VCdim}(H) = \max\{m : \Pi_H(m) = 2^m\},$$

where we take the maximum to be infinite if the set is unbounded.

A result which is often useful is that if H is a finite hypothesis space then H has VC dimension at most $\log |H|$; this follows from the observation that if d is the VC dimension of H and $\mathbf{x} \in X^d$ is shattered by H, then $|H| \geq |H|_{\mathbf{x}}| = 2^d$. (Here, and throughout, log denotes logarithm to base 2.)

Consider now the perceptron P_n with n inputs. The set of positive examples of the function h_ω computed by the perceptron in state $\omega = (\alpha_1, \alpha_2, \ldots, \alpha_n, \theta)$ is the closed half-space l_ω^+ consisting of $y \in \mathbf{R}^n$ such that $\sum_{i=1}^n \alpha_i y_i \geq \theta$. This is bounded by the *hyperplane* l_ω with equation $\sum_{i=1}^n \alpha_i y_i = \theta$. Roughly speaking, l_ω divides $\mathbf{R}^n$ into the set of positive examples of h_ω and the set of negative examples of h_ω

We shall use the following result, known as *Radon's Theorem*, in which, for $S \subseteq \mathbf{R}^n$, $\text{conv}(S)$ denotes the convex hull of S. Let n be any positive integer, and let E be any set of $n+2$ points in $\mathbf{R}^n$. Then there is a non-empty subset S of E such that

$$\text{conv}(S) \cap \text{conv}(E \setminus S) \neq \emptyset.$$

A proof is given, for example, by Grunbaum (1967).

Theorem 4 *For any positive integer n, let P_n be the real perceptron with n inputs. Then*

$$\text{VCdim}(P_n) = n + 1.$$

Proof Let $\mathbf{x} = (x_1, x_2, \ldots, x_{n+2})$ be any sample of length $n+2$. As we have noted, if two of the examples are equal then $\mathbf{x}$ cannot be shattered. Suppose then that the set $E_{\mathbf{x}}$ of examples in $\mathbf{x}$ consists of $n+2$ distinct points in $\mathbf{R}^n$. By Radon's Theorem, there is a non-empty subset S of $E_{\mathbf{x}}$ such that $\text{conv}(S) \cap \text{conv}(E_{\mathbf{x}} \setminus S) \neq \emptyset$. Suppose that there is a hypothesis h_ω in P_n such that S is the set of positive examples of h_ω in $E_{\mathbf{x}}$. Then we have

$$S \subseteq l_\omega^+, \quad E_{\mathbf{x}} \setminus S \subseteq \mathbf{R}^n \setminus l_\omega^+.$$

Since open and closed half-spaces are convex subsets of $\mathbf{R}^n$, we also have

$$\text{conv}(S) \subseteq l_\omega^+, \quad \text{conv}(E_{\mathbf{x}} \setminus S) \subseteq \mathbf{R}^n \setminus l_\omega^+.$$

Therefore

$$\text{conv}(S) \cap \text{conv}(E_{\mathbf{x}} \setminus S) \subseteq l_\omega^+ \cap \mathbf{R}^n \setminus l_\omega^+ = \emptyset,$$

which is a contradiction. We deduce that no such h_ω exists and therefore that $\mathbf{x}$ is not shattered by P_n. Thus *no* sample of length $n+2$ is shattered by P_n and the VC dimension of P_n is at most $n+1$.

It remains to prove the reverse inequality. Let o denote the origin of $\mathbf{R}^n$ and, for $1 \leq i \leq n$, let e_i be the point with a 1 in the ith coordinate and all other coordinates 0.

Then P_n shatters the sample $\mathbf{x} = (o, e_1, e_2, \ldots, e_n)$ of length $n+1$. To see this, suppose that S is a subset of $E_{\mathbf{x}} = \{o, e_1, \ldots, e_n\}$. For $i = 1, 2, \ldots, n$, let α_i be 1 if $e_i \in S$ and -1 otherwise, and let θ be $-1/2$ if $o \in S$, $1/2$ otherwise. Then it is straightforward to verify that if ω is the state $\omega = (\alpha_1, \alpha_2, \ldots, \alpha_n, \theta)$ of P_n then the set of positive examples of h_ω in $E_{\mathbf{x}}$ is precisely S. Therefore $\mathbf{x}$ is shattered by P_n and, consequently, $\mathrm{VCdim}(P_n) \geq n+1$. $\square$

The growth function $\Pi_H(m)$ of a hypothesis space of finite VC dimension is a measure of how many different classifications of an m-sample into positive and negative examples can be achieved by the hypotheses of H, while the VC dimension of H is the maximum value of m for which $\Pi_H(m) = 2^m$. Clearly these two quantities are related, because the VC dimension is defined in terms of the growth function. But there is another, less obvious, relationship: the growth function $\Pi_H(m)$ can be bounded by a polynomial function of m, and the degree of the polynomial is the VC dimension d of H. Explicitly, we have the following theorem. The first inequality is due to Sauer (1972) and is usually known as *Sauer's Lemma*. The second inequality is elementary—a proof was given by Blumer *et al.* (1989).

Theorem 5 (Sauer's Lemma) *Let $d \geq 0$ and $m \geq 1$ be given integers and let H be a hypothesis space with* $\mathrm{VCdim}(H) = d \geq 1$. *Then for $m \geq d$,*

$$\Pi_H(m) \leq \Phi(d, m) = \sum_{i=0}^{d} \binom{m}{i} < \left(\frac{em}{d}\right)^d,$$

where e is the base of natural logarithms. $\square$

We have motivated our discussion of VC dimension by describing it as a measure of the expressive power of a hypothesis space. We shall see that it turns out to be a key parameter for quantifying the difficulty of pac learning. Our first result along these lines is that finite VC dimension is necessary for potential learnability.

Theorem 6 *If a hypothesis space has infinite VC dimension then it is not potentially learnable.*

Proof Suppose that H has infinite VC dimension, so that for any positive integer m there is a sample $\mathbf{z}$ of length $2m$ which is shattered by H. Let $E = E_{\mathbf{z}}$ be the set of examples in this sample and define a probability distribution μ on X by $\mu(x) = 1/2m$ if $x \in E$ and $\mu(x) = 0$ otherwise. In other words, μ is uniform on E and zero elsewhere. We observe that μ^m is uniform on E^m and zero elsewhere. Thus, with probability one, a randomly chosen sample $\mathbf{x}$ of length m is a sample of examples from E. Let $\mathbf{s} = (\mathbf{x}, t(\mathbf{x})) \in S(m, t)$ be a training sample of length m for a target concept $t \in H$. With probability 1 (with respect to μ^m), we have $x_i \in E$ for $1 \leq i \leq m$. Since $\mathbf{z}$ is shattered by H, there is a hypothesis $h \in H$ such that $h(x_i) = t(x_i)$ for each

x_i ($1 \leq i \leq m$), and $h(x) \neq t(x)$ for all other x in E. It follows that h is consistent with $\mathbf{s}$, whereas h has error at least $1/2$ with respect to t. We have shown that for any positive integer m, and any target concept t, there is a probability distribution μ on X such that the set

$$\{\mathbf{s} \mid \text{for all } h \in H,\ \mathrm{er}_{\mathbf{s}}(h) = 0 \Longrightarrow \mathrm{er}_{\mu}(h) < 1/2\}$$

has probability zero. Thus, H is not potentially learnable. □

The converse of the preceding theorem is also true: finite VC dimension is sufficient for potential learnability. This result can be traced back to the statistical researches of Vapnik and Chervonenkis (1971) (see also Vapnik (1982) and Vapnik and Chervonenkis (1981)). The work of Blumer *et al.* (1989) showed that it is one of the key results in Computational Learning Theory. We now give some indication of its proof.

Suppose that the hypothesis space H is defined on the example space X, and let t be any target concept in H, μ any probability distribution on X and ϵ any real number with $0 < \epsilon < 1$. The objects t, μ, ϵ are to be thought of as fixed, but arbitrary, in what follows. The probability of choosing a training sample for which there is a consistent, but ϵ-bad, hypothesis is

$$\mu^m \left\{\mathbf{s} \in S(m,t) \,|\, \text{there is } h \in H \text{ such that } \mathrm{er}_{\mathbf{s}}(h) = 0, \mathrm{er}_{\mu}(h) \geq \epsilon\right\}.$$

Thus, in order to show that H is potentially learnable, it suffices to find an upper bound $f(m,\epsilon)$ for this probability which is independent of both t and μ and which tends to 0 as m tends to infinity.

The following result, of the form just described, is due to Blumer *et al.* (1989), and generalises a result of Haussler and Welzl (1987). Better bounds have subsequently been obtained by Anthony, Biggs and Shawe-Taylor (1990) (see also Shawe-Taylor, Anthony and Biggs (1993)), but the result presented here suffices for the present discussion.

Theorem 7 *Suppose that H is a hypothesis space defined on an example space X, and that t, μ, and ϵ are arbitrary, but fixed. Then*

$$\mu^m \left\{\mathbf{s} \in S(m,t) \,|\, \text{there is } h \in H \text{ such that } \mathrm{er}_{\mathbf{s}}(h) = 0, \mathrm{er}_{\mu}(h) \geq \epsilon\right\} < 2\,\Pi_H(2m)\,2^{-\epsilon m/2}$$

for all positive integers $m \geq 8/\epsilon$. □

The right-hand side is the bound $f(m,\epsilon)$ as postulated above. If H has finite VC dimension then, by Sauer's Lemma, $\Pi_H(2m)$ is bounded by a polynomial function of m, and therefore $f(m,\epsilon)$ is eventually dominated by the negative exponential term. Thus the right-hand side, which is independent of t and μ, tends to 0 as m tends to infinity and, by the above discussion, this establishes potential learnability for spaces of finite VC dimension.

At this point it is helpful to introduce a new piece of terminology. Supose that real numbers $0 < \delta, \epsilon < 1$ are given, and let L be a learning algorithm for a concept space C and a hypothesis space H. We say that the *sample complexity* of L on $T \subseteq C$ is the least value $m_L(T, \delta, \epsilon)$ such that, for all target concepts $t \in T$ and all probability distributions μ,

$$\mu^m\{\mathbf{s} \in S(m,t) \mid \mathrm{er}_\mu(L(\mathbf{s})) < \epsilon\} > 1 - \delta$$

whenever $m \geq m_L(T, \delta, \epsilon)$; in other words, a sample of length $m_L(T, \delta, \epsilon)$ is sufficient to ensure that the output hypothesis $L(\mathbf{s})$ is pac, with the given values of δ and ϵ. In practice we often omit T when this is clear and we usually deal with a convenient upper bound $m_0 \geq m_L$, rather than m_L itself; thus $m_0(\delta, \epsilon)$ will denote *any* value sufficient to ensure that the pac conclusion, as stated above, holds for all $m \geq m_0$.

The following result follows from Theorem 7.

Theorem 8 *There is a constant K such that if hypothesis space H has VC dimension $d \geq 1$ and the concept space C is a subset of H, then any consistent learning algorithm L for (C, H) is pac, with sample complexity*

$$m_L(\delta, \epsilon) \leq \frac{K}{\epsilon}\left(d \ln\left(\frac{1}{\epsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right),$$

for $0 < \delta, \epsilon < 1$. □

Values of K are easily obtained; see Blumer *et al.* (1989), Anthony, Biggs and Shawe-Taylor (1990) and Anthony and Biggs (1992). This result provides an extension of the bound for finite spaces, mentioned at the beginning of the section, to spaces with finite VC dimension.

Example The real perceptron P_n has VC dimension $n + 1$. Suppose that for any training sample for a hypothesis of P_n, we can find a state ω of the perceptron such that h_ω is consistent with the sample. Then if we use a training sample of length

$$\left\lceil \frac{K}{\epsilon}\left((n+1) \ln\left(\frac{1}{\epsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right)\right\rceil,$$

we are guaranteed a probably approximately correct output hypothesis, regardless of both the target hypothesis and the probability distribution on the examples. □

We now present a lower bound result, part of which is due to Ehrenfeucht *et al.* (1989) and the other part of which is due to Blumer *et al.* (1989). This provides a lower bound on the sample complexity of *any* (C, H) pac learning algorithm when C has finite VC dimension and is 'non-trivial'.

Theorem 9 *Let C be a concept space and H a hypothesis space, such that C has VC dimension at least 1 and consists of at least three distinct concepts. Suppose that L is* *any pac learning algorithm for (C, H). Then the sample complexity of L satisfies*

$$m_L(\delta, \epsilon) > \max\left(\frac{\mathrm{VCdim}(C) - 1}{32\epsilon}, \frac{1}{\epsilon}\ln\left(\frac{1}{\delta}\right)\right),$$

for all $\epsilon \leq 1/8$ and $\delta \leq 1/100$. $\square$

An immediate corollary of this result is that if a concept space C has infinite VC dimension then there is no pac learning algorithm for (C, H) for any hypothesis space H.

An interesting consequence of these results is that a set H of functions is pac learnable if and only if it is potentially learnable, since each condition holds precisely when H has finite VC dimension. The results show that if there is a pac learning algorithm for H, then *any* consistent learning algorithm for H is pac.

4. FEEDFORWARD ARTIFICIAL NEURAL NETWORKS

A perceptron contains only one 'active unit', and is consequently severely limited in its capabilities. The idea that more complex assemblies of units may have greater power is an old one, motivated by the fact that living brains seem to be constructed in this way, and it has led to the intensive study of 'artificial neural networks'. We shall examine some such networks in the context of Computational Learning Theory.

The basic structure is a pair of sets (N, A), where N is a finite set whose members are called *nodes*, and A is a subset of $N \times N$ whose members are called *arcs*. The structure (N, A) is a *directed graph*, or *digraph*, which we think of as a fixed architecture for a 'machine'. For simplicity, we consider only digraphs which have no directed cycles: that is, there is no sequence of arcs beginning with (r, s) and ending with (q, r), for any node r. In the present context this is known as the *feedforward* condition.

In order to present this set-up as a 'machine', we require some additional features. First we specify a subset J of the nodes, which we call *input nodes*, and a single node $z \notin J$ which we call the *output node*. The underlying idea is that all nodes receive and transmit signals; the input nodes receive their signals from the outside world and the output node transmits a signal to the outside world, while all other nodes receive and transmit along the relevant arcs of the digraph. Each arc (r, s) has a *weight*, $w(r, s)$, which is a real number representing the strength of the connection between the nodes r and s. A positive weight corresponds to an 'excitatory' connection, a negative weight to an 'inhibitory' connection. Another feature is that all nodes except the input nodes are 'active', in that they transmit a signal which is a predetermined function of the signals they receive. For this reason, the nodes in $N \setminus J$ are called *computation nodes*. To make this idea precise, we introduce an *activation function* f_r for each computation node r. The activity of such a node is specified in two stages. First the signals arriving at r are aggregated by taking their weighted sum according to the connection strengths on the arcs with terminal node r, and then the function f_r of this value is computed. Thus the action of the entire network may be described in terms of two functions $p\colon N \to \mathbf{R}$ and $q\colon N \to \mathbf{R}$, representing the received and transmitted signals respectively. We assume that a vector of real-valued signals $y = (y_j)_{j\in J}$ is applied externally to the input nodes, and $p(j) = q(j) = y_j$ for each j in J. For each computation node l the received and transmitted signals are defined as follows.

$$p(l) = \sum_{\{i|(i,l)\in A\}} q(i)w(i,l), \qquad\qquad q(l) = f_l(p(l)).$$

The output is the value $q(z)$ transmitted by the node z.

For now, we assume that every activation function is a simple linear threshold function: $f(u) = 1$ if $u \geq \theta$ and $f(u) = 0$ otherwise. We shall write θ_r to denote the value of the threshold for the node r.

When all the computation nodes are linear threshold nodes, a state ω of the machine is described by the real numbers $w(r,s)$ and θ_r, for $(r,s) \in A, r \in N \setminus J$. The set of all states which satisfy some given rules (such as bounds on the values of the weights and thresholds) will be denoted by Ω. Now we are firmly within the framework developed earlier. The function computed by the machine in state ω will be denoted by h_ω, so that $h_\omega(y) = q(z)$. Note that this is a boolean value, because the output node has a linear threshold activation function. The set $\{h_\omega \mid \omega \in \Omega\}$ of functions computable by the machine is the hypothesis space H, and the assignment $\omega \mapsto h_\omega$ is a representation $\Omega \to H$.

5. EXPRESSIVE POWER OF THRESHOLD NETWORKS

We shall prove a result of Baum and Haussler (1989), which gives an upper bound on the VC dimension of a feedforward linear threshold network in terms of the number of nodes and arcs.

Suppose that we have a feedforward network of linear threshold nodes, with underlying digraph (N, A) and set of states Ω. The feedforward condition allows us to label the computation nodes by the positive integers in their natural order, $1, 2, \ldots, z$, in such a way that z is the output node and $(i,j) \in A$ implies $j > i$. (This may be done by numbering first those computation nodes which are linked only to input nodes, then those which are linked only to input nodes and already-numbered computation nodes, and so on.)

For each state $\omega \in \Omega$, corresponding to an assignment of weights and thresholds to all the arcs and computation nodes, we let ω^l denote the part of ω determined by the thresholds on computation nodes $1, 2, \ldots, l$ and the weights on arcs which terminate at those nodes. Then for $2 \leq l \leq z$ we have the decomposition $\omega^l = (\omega^{l-1}, \zeta_l)$ where ζ_l stands for the weights on arcs terminating at l and the threshold at l. In isolation, the output of a computation node l is a linear threshold function, determined by ζ_l, of the outputs of all those nodes j for which (j, l) is an arc; some of these may be input nodes and some may be computation nodes with $j < l$. We denote the space of such functions by H_l and the growth function of this 'local hypothesis space' by Π_l.

Suppose that $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is a sample of inputs to the network. (Each example x_i is a $|J|$-vector of real numbers, where J is the set of input nodes.) For any computation node l $(1 \leq l \leq z)$, we shall say that states ω_1, ω_2 of the network are *l-distinguishable by* $\mathbf{x}$ if the following holds. There is an example in $\mathbf{x}$ such that, when this example is input, the output of at least one of the computation nodes $1, 2, \ldots, l$, is different when the state is ω_1 from its output when the state is ω_2. In other words, if one has access to the signals transmitted by nodes 1 to l only, then, using the sample $\mathbf{x}$, one can differentiate between the two states. We shall denote by $S_l(\mathbf{x})$ the number of different states which are mutually l-distinguishable by $\mathbf{x}$.

Lemma 10 *With the notation defined as above, we have*

$$S_l(\mathbf{x}) \leq \Pi_1(m)\,\Pi_2(m)\ldots\Pi_l(m), \quad (1 \leq l \leq z).$$

Proof We prove the claim by induction on l. For $l = 1$ we have $S_1(\mathbf{x}) \leq \Pi_1(\mathbf{x})$, because two states are 1-distinguishable if and only if they give different classifications of the training sample at node 1. Thus $S_1(\mathbf{x}) \leq \Pi_1(m)$. Assume, inductively, that the claim holds for $l = k - 1$, where $2 \leq k \leq z$. The decomposition $\omega^k = (\omega^{k-1}, \zeta_k)$ shows that if two states are k-distinguishable but not $(k-1)$-distinguishable, then they must be distinguished by the action of the node k. For each of the $S_{k-1}(\mathbf{x})$ $(k-1)$-distinguishable states there are thus at most $\Pi_k(m)$ k-distinguishable states. Hence $S_k(\mathbf{x}) \leq S_{k-1}(\mathbf{x})\,\Pi_k(m)$. By the inductive assumption, the right-hand side is at most $\Pi_1(m)\,\Pi_2(m)\ldots\Pi_k(m)$. The result follows. $\square$

If H is the hypothesis space of N then $\Pi_H(\mathbf{x})$ is the number of states which are mutually z-distinguishable by $\mathbf{x}$. Thus,

$$\Pi_H(m) \leq \Pi_1(m)\,\Pi_2(m)\ldots\Pi_z(m),$$

for any positive integer m. The next result follows from this observation and the previous result.

Corollary 11 *Let (N, A) be a feedforward linear threshold network with z computation nodes, and let $W = |N \setminus J| + |A|$ be the total number of variable weights and thresholds. Let H be the hypothesis space of the network. Then for $m > W$, we have*

$$\Pi_H(m) \leq \left(\frac{zem}{W}\right)^W.$$

Proof Certainly, $W \geq d(i) + 1$ for $1 \leq i \leq z$ and so, for each such i and for $m > W$, $\Pi_i(m) \leq (em/d(i)+1)^{d(i)+1}$, by Sauer's Lemma and since the VC dimension of H_i is

$d(i)+1$. It follows that

$$\Pi_H(m) \le \Pi_1(m)\,\Pi_2(m)\ldots\Pi_z(m) \le \prod_{i=1}^{z}\left(\frac{em}{d(i)+1}\right)^{d(i)+1}.$$

From this one can obtain the desired result. We omit the details here; these may be found in Baum and Haussler (1989) or Anthony and Biggs (1992). □

Theorem 12 *The VC dimension of a feedforward linear threshold network with z computation nodes and a total of W variable weights and thresholds is at most $2W\log(ez)$.*

Proof Let H be the hypothesis space of the network. By the above result, we have, for $m \ge W$, $\Pi_H(m) \le (zem/W)^W$, where W is the total number of weights and thresholds. Now,

$$\left(\frac{2ezW\log(ez)}{W}\right)^W < 2^{2W\log(ez)} \iff 2ez\log(ez) < (ez)^2 \iff 2\log(ez) < ez,$$

which is true for any $z \ge 1$. Therefore, $\Pi_H(m) < 2^m$ when $m = 2W\log(ez)$, and the VC dimension of H is at most $2W\log(ez)$, as claimed. □

Notice that the upper bound on the VC dimension depends only on the 'size' of the network; that is, on the number of computation nodes and the number of arcs. That it is independent of the structure of the network — the underlying directed graph — suggests that it may not be a very tight bound. In their paper, Baum and Haussler (1989) showed that certain simple networks have VC dimension at least a constant multiple of the number of weights. More recently, Bartlett (1992) obtained similar results for wider classes of networks. However, in a result which shows that the upper bound is essentially the best that can be obtained, Maass (1992) has shown that there is a constant c such that for infinitely many values of W, some feedforward linear threshold network with W weights has VC dimension at least $cW\log W$. (The networks for which Maass showed this to be true have 4 layers.)

If, as in Baum and Haussler (1989), we substitute the bound of Corollary 11 directly into the result of Theorem 7 then we can derive a better upper bound on sample complexity than would result from substituting the VC dimension bound into Theorem 8. Indeed, the former method gives a bound involving a $\log(z/\epsilon)$ term, while the latter yields a bound depending on $\log z \log(1/\epsilon)$. With this observation and the previous results, we have the following result on sufficient sample size.

Theorem 13 *Let (N, A) be a feedforward linear threshold network having z computation nodes and W variable weights and thresholds. Then for all $0 < \delta, \epsilon < 1$, there is $m_0(\delta, \epsilon)$ such that if a training sample of length $m \ge m_0(\delta, \epsilon)$ is drawn at random according to some fixed probability distribution on the set of all inputs, then the following* *holds. With probability at least $1-\delta$, if the sample is 'loaded' onto the network, so that the function computed by the network is consistent with the sample, then the network correctly classifies a further randomly chosen input with probability at least $1-\epsilon$. The sufficient sample size satisfies*

$$m_0(\delta,\epsilon) \le \frac{K_1}{\epsilon}\left(W \log\left(\frac{z}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right)$$

for some (absolute) constant K_1. Furthermore, there is $K_2 > 0$ such that for infinitely many W, there is a network with W weights for which the sufficient sample size must satisfy

$$m_0(\delta,\epsilon) \ge \frac{K_2}{\epsilon}\left(W \log W + \log\left(\frac{1}{\delta}\right)\right).$$

□

6. THE COMPUTATIONAL COMPLEXITY OF LEARNING

Thus far, a learning algorithm has been defined as a function mapping training samples into hypotheses. We shall now be more specific about the *algorithmics*. If pac learning by a learning algorithm is to be of practical value, it must, first, be possible to implement the learning algorithm on a computer; that is, it must be *computable* and therefore, in a real sense, an *algorithm*, not just a function. Further, it should be possible to implement the algorithm 'quickly'.

The subject known as *Complexity Theory* deals with the relationship between the size of the input to an algorithm and the time required for the algorithm to produce its output for an input of that size. In particular, it is concerned with the question of when this relationship is such that the algorithm can be described as 'efficient'. Here, we shall describe the basic ideas in a very simplistic way. More details may be found in the books by Garey and Johnson (1979), Wilf (1986), and Cormen, Leiserson and Rivest (1990).

The *size* of an input to an algorithm will be denoted by s. For example, if an algorithm has a binary encoding as input, the size of an input could be the number of bits it contains. Equally, if the input is a real vector, one could define the size to be the dimension of the vector. Let A be an algorithm which accepts inputs of varying size s. We say that *the running time of A is $O(f(s))$* if there is some constant K such that, for any input of size s, the number of operations required to produce the output of A is at most $Kf(s)$. Note that this definition is 'device-independent' because the running time depends only on the *number* of operations carried out, and not on the actual speed with which such an operation can be performed. Furthermore, the running time is a *worst-case* measure; we consider the maximum possible number of operations taken over all inputs of a given size.

There are good reasons for saying that an algorithm with running time $O(s^r)$, for some fixed integer $r \ge 1$, is 'efficient'. Such an algorithm is said to be a *polynomial time* algorithm, and problems which can be solved by a polynomial time learning algorithm are usually regarded as 'easy'. Thus, to show that a problem is easy, we should present a polynomial time algorithm for it. On the other hand, if we wish to show that a given problem is 'hard', it is enough to show that if this problem could be solved in polynomial time then so too could another problem which is believed to be hard. One standard problem which is believed to be hard is the *graph k-colouring problem* for $k \geq 3$. Let G be a graph with vertex-set V and edge-set E, so that E is a subset of the set of 2-element subsets of V. A *k-colouring* of G is a function $\chi : V \to \{1, 2, \ldots, k\}$ with the property that, whenever $ij \in E$, then $\chi(i) \neq \chi(j)$. The graph k-colouring problem may formally be stated as:

> *GRAPH k-COLOURING*
> **Instance** A graph $G = (V, E)$.
> **Question** Is there a k-colouring of G?

When we say that *GRAPH k-COLOURING* is 'believed to be hard', we mean that it belongs to a class of problems known as the *NP-complete* problems. This class of problems is very extensive, and contains many famous problems in Discrete Mathematics. Although it has not yet been proved, it is conjectured, and widely believed, that there is no polynomial time algorithm for any of the NP-complete problems. This is known as the 'P $\neq$ NP conjecture'.

We shall apply these ideas in the following way. Suppose that Π is a problem in which we are interested, and Π_0 is a problem which is known to be NP-complete. Suppose also that we can demonstrate that if there is a polynomial time algorithm for Π then there is one for Π_0. In that case our problem Π is said to be *NP-hard*. If the P $\neq$ NP conjecture is true, then proving that a problem Π is NP-hard establishes that there is no polynomial time algorithm for Π.

We now wish to quantify the behaviour of learning algorithms with respect to n, and it is convenient to make the following definitions. We say that a union of hypothesis spaces $H = \bigcup H_n$ is *graded* by example size n, when H_n denotes the space of hypotheses defined on examples of size n. For example, H_n may be the space P_n of the perceptron, defined on real vectors of length n. By a *learning algorithm for $H = \bigcup H_n$* we mean a function L from the set of training samples for hypotheses in H to the space H, such that when $\mathbf{s}$ is a training sample for $h \in H_n$ it follows that $L(\mathbf{s}) \in H_n$. That is, we insist that L preserves the grading. (Analogously, one may define, more generally, a learning algorithm for (C, H) when each of C and H are graded.) An example of a learning algorithm defined on the graded perceptron space $P = \bigcup P_n$ is the perceptron learning algorithm of Rosenblatt (1959). (See also Minsky and Papert (1969).) Observe that this algorithm acts in essentially the same manner on each P_n; the 'rule' is the same for each n.

Consider a learning algorithm L for a hypothesis space $H = \bigcup H_n$, graded by example size. An input to L is a training sample, which consists of m examples of size n together with the m single-bit labels. The total size of the input is therefore $m(n + 1)$, and it would be possible to use this single number as the measure of input size. However, there is some advantage in keeping track of m and n separately, and so we shall use the notation $R_L(m,n)$ to denote the worst-case running time of L on a training sample of m examples of size n.

A learning algorithm L for $\bigcup H_n$ is said to be a pac learning algorithm if L acts as a pac learning algorithm for each H_n. The sample complexity provides the link between the running time $R_L(m,n)$ of a learning algorithm (that is, the number of operations required to produce its output on a sample of length m when the examples have size n) and its running time as a pac learning algorithm (that is, the number of operations required to produce an output which is probably approximately correct with given parameters). Since a sample of length $m_0(H_n,\delta,\epsilon)$ is sufficient for the pac property, the number of operations required is at most $R_L(m_0(H_n,\delta,\epsilon),n)$.

Until now, we have regarded the accuracy parameter ϵ as fixed but arbitrary. It is clear that decreasing this parameter makes the learning task more difficult, and therefore the running time of an efficient pac learning algorithm should be constrained in some appropriate way as ϵ^{-1} increases. We say that a learning algorithm L for $H=\bigcup H_n$ is *efficient with respect to accuracy and example size* if its running time is polynomial in m and the sample complexity $m_L(H_n,\delta,\epsilon)$ depends polynomially on n and ϵ^{-1}.

We are now ready to consider the implications for learning of the theory of NP-hard problems. Let $H=\bigcup H_n$ be a hypothesis space of functions, graded by the example size n. The *consistency problem* for H may be stated as follows.

H−CONSISTENCY
Instance A training sample $\mathbf{s}$ of labelled examples of size n.
Question Is there a hypothesis in H_n consistent with $\mathbf{s}$?

In practice, we wish to produce a consistent hypothesis, rather than simply know whether or not one exists. In other words, we have to solve a 'search' problem, rather than an 'existence' problem. But these problems are directly related. Suppose that we consider only those $\mathbf{s}$ with length bounded by some polynomial in n. Then, if we can find a consistent hypothesis in time polynomial in n, we can answer the existence question by the following procedure. Run the search algorithm for the time (polynomial in n) in which it is guaranteed to find a consistent hypothesis if there is one; then check the output hypothesis explicitly against the examples in $\mathbf{s}$ to determine whether or not it is consistent. This checking can be done in time polynomial in n also. Thus if we can show that a restricted form of the existence problem is NP-hard, this means that there is no polynomial time algorithm for the corresponding search problem (unless P = NP).

If there is a consistent learning algorithm L for a graded hypothesis space $H=\bigcup H_n$ such that VCdim(H_n) is polynomial in n and the algorithm runs in time polynomial in the sample length m, then the results presented earlier show that L pac learns H_n with running time polynomial in n and ϵ^{-1} and so is efficient with respect to accuracy and example size. Roughly speaking we may say that an efficient 'consistent-hypothesisfinder' is an efficient 'pac learner'. It is natural to ask to what extent the converse is true. It turns out that efficient pac learning does imply efficient consistent-hypothesis-finding, provided we are prepared to accept a *randomised algorithm.* A full account of the meaning of this term may be found in the book of Cormen, Leiserson and Rivest (1990), but for our purposes the idea can be explained in a few paragraphs.

We suppose that there is available some form of *random number generator* which, given any integer $I \geq 2$, produces a stream of integers i in the range $1 \leq i \leq I$, each particular value being equally likely. This could be done electronically, or by tossing an I-sided die. A randomised algorithm A is allowed to use these random numbers as part of its input. The computation carried out by the algorithm is determined by its input, so that it depends on the particular sequence produced by the random number generator. It follows that we can speak of the probability that A has a given outcome, by which is meant the proportion of sequences which produce that outcome.

We say that a randomised algorithm A 'solves' a search problem Π if it behaves in the following way. The algorithm always halts and produces an output. If A has failed to find a solution to Π then the output is simply *no.* But, with probability at least $1/2$ (in the sense explained above), A succeeds in finding a solution to Π and its output is this solution. The practical usefulness of a randomised algorithm stems from the fact that repeating the algorithm several times dramatically increases the likelihood of success. If the algorithm fails at the first attempt, which happens with probability at most $1/2$, then we simply try again. The probability that it fails twice in succession is at most $1/4$. Similarly, the probability that it fails in k attempts is at most $(1/2)^k$, which approaches zero very rapidly with increasing k. Thus in practice a randomised algorithm is almost as good as an ordinary one — provided of course that it has polynomial running time. We have the following theorem of Pitt and Valiant (1988) (see also Natarajan (1989) and Haussler *et al.* (1988)).

Theorem 14 *Let $H = \bigcup H_n$ be a hypothesis space and suppose that there is a pac learning algorithm for H which is efficient with respect to accuracy and example size. Then there is a randomised algorithm which solves the problem of finding a hypothesis in H_n consistent with a given training sample of a hypothesis in H_n, and which has running time polynomial in n and m (the length of the training sample).*

Proof Suppose that $\mathbf{s}^*$ is a training sample for a target hypothesis $t \in H_n$, and that $\mathbf{s}^*$ contains m^* distinct labelled examples. We shall show that it is possible to find a hypothesis consistent with $\mathbf{s}^*$ by running the given pac learning algorithm L on a related training sample. Define a probability distribution μ on the example space X by $\mu(x) = 1/m^*$ if x occurs in $\mathbf{s}^*$ and $\mu(x) = 0$ otherwise. We can use a random number generator with output values i in the range 1 to m^* to select an example from X according to this distribution: simply regard each random number as the label of one of the m^* equiprobable examples. Thus the selection of a training sample of length m for t, according to the probability distribution μ, can be simulated by generating a sequence of m random numbers in the required range.

Let L be a pac learning algorithm as postulated in the statement of the Theorem. Then, when δ, ϵ, are given, we can find an integer $m_0(n, \delta, \epsilon)$ for which the probability (with respect to training samples $\mathbf{s} \in S(m_0, t)$) that the error of $L(\mathbf{s})$ is less than ϵ is greater than $1 - \delta$. Suppose we specify the confidence and accuracy parameters to be $\delta = 1/2$ and $\epsilon = 1/m^*$. Then if we run the given algorithm L on a training sample of length $m_0(n, 1/2, 1/m^*)$, drawn randomly according to the distribution μ, the pac property of L ensures that the probability that the error of the output is less than $1/m^*$ is greater than $1-1/2 = 1/2$. Since there are no examples with probability strictly between 0 and $1/m^*$, this implies that the probability that the output agrees exactly with the training sample is greater than $1/2$.

The procedure described in the previous paragraph is the basis for a randomised algorithm L^* for finding a hypothesis which agrees with the given training sample $\mathbf{s}^*$. In summary, L^* consists of the following steps.

- Evaluate $m_0 = m_0(n, 1/2, 1/m^*)$.
- Construct, as described, a sample $\mathbf{s}$ of length m_0, according to μ.
- Run the given pac learning algorithm L on $\mathbf{s}$.
- Check $L(\mathbf{s})$ explicitly to determine whether or not it agrees with $\mathbf{s}^*$.
- If $L(\mathbf{s})$ does not agree with $\mathbf{s}^*$, output *no*. If it does, output $L(\mathbf{s})$.

As we noted, the pac property of L ensures that L^* succeeds with probability greater than $\frac{1}{2}$. Finally, it is clear that, since the running time of L is polynomial in m and its sample complexity $m_0(n, 1/2, 1/m^*)$ is polynomial in n and $m^* = 1/\epsilon$, the running time of L^* is polynomial in n and m^*. □

7. HARDNESS RESULTS FOR NEURAL NETWORKS

The fact that computational complexity-theoretic hardness results hold for neural networks was first shown by Judd (1988). In this section we shall prove a simple hardness result along the lines of one due to Blum and Rivest (1988).

The machine has n input nodes and $k+1$ computation nodes ($k \geq 1$). The first k computation nodes are 'in parallel' and each of them is connected to all the input nodes. The last computation node is the output node; it is connected by arcs with fixed weight 1 to the other computation nodes, and it has fixed threshold k. The effect of this arrangement is that the output node acts as a multiple AND gate for the outputs of the other computation nodes. We shall refer to this machine (or its hypothesis space) as P_n^k.

A state ω of P_n^k is described by the thresholds θ_l ($1 \leq l \leq k$) of the first k computation nodes and the weights $w(i, l)$ on the arcs (i, l) linking the input nodes to the computation nodes. We shall use the notation $\alpha^{(l)}$ for the n-vector of weights on the arcs terminating at l, so that $\alpha_i^{(l)} = w(i, l)$. The set Ω of such states provides a representation $\Omega \rightarrow P_n^k$

in the usual way.

We shall prove that the consistency problem for $P^k = \bigcup P_n^k$ is NP-hard (provided $k \geq 3$), by reducing the problem to *GRAPH k-COLOURING*. Let G be a graph with vertex-set $V = \{1, 2, \ldots, n\}$ and edge-set E. We construct a training sample $\mathbf{s}(G)$ as follows. For each vertex $i \in V$ we take as a negative example the vector v_i which has 1 in the ith coordinate position, and 0's elsewhere. For each edge $ij \in E$ we take as a positive example the vector $v_i + v_j$. We also take the zero vector $o = 00\ldots0$ to be a positive example.

Theorem 15 *There is a function in P_n^k which is consistent with $\mathbf{s}(G)$ if and only if the graph G is k-colourable.*

Proof Suppose $h \in P_n^k$ is consistent with the training sample. By the construction of the network, h is a conjunction $h = h_1 \wedge h_2 \wedge \ldots \wedge h_k$ of linear threshold functions. (That is, $h(x) = 1$ if and only if $h_i(x) = 1$ for all i between 1 and k.) Specifically, there are weight-vectors $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(k)}$ and thresholds $\theta_1, \theta_2, \ldots, \theta_k$ such that

$$h_l(y) = 1 \iff \langle \alpha^{(l)}, y \rangle \geq \theta_l \ \ (1 \leq l \leq k).$$

Note that, since o is a positive example, we have $0 = \langle \alpha^{(l)}, o \rangle \ \geq \ \theta_l$ for each l between 1 and k. For each vertex i, $h(v_i) = 0$, and so there is at least one function h_f $(1 \leq f \leq k)$ for which $h_f(v_i) = 0$. Thus we may define $\chi : V \to \{1, 2, \ldots, k\}$ by

$$\chi(i) = \min\{f \mid h_f(v_i) = 0\}.$$

It remains to prove that χ is a colouring of G. Suppose that $\chi(i) = \chi(j) = f$, so that $h_f(v_i) = h_f(v_j) = 0$. In other words,

$$\langle \alpha^{(f)}, v_i \rangle < \theta_f, \ \ \langle \alpha^{(f)}, v_j \rangle < \theta_f.$$

Then, recalling that $\theta_f \leq 0$, we have

$$\langle \alpha^{(f)}, v_i + v_j \rangle < \theta_f + \theta_f \leq \theta_f.$$

It follows that $h_f(v_i + v_j) = 0$ and $h(v_i + v_j) = 0$. Now if ij were an edge of G, then we should have $h(v_i + v_j) = 1$, because we assumed that h is consistent with the training sample. Thus ij is not an edge of G, and χ is a colouring, as claimed.

Conversely, suppose we are given a colouring $\chi : V \to \{1, 2, \ldots, k\}$. For $1 \leq l \leq k$ define the weight-vector $\alpha^{(l)}$ as follows: $\alpha_i^{(l)} = -1$ if $\chi(i) = l$ and $\alpha_i^{(l)} = 1$ otherwise. Define the threshold θ_l to be $-1/2$. Let $h_1, h_2, \ldots, h_k$ be the corresponding linear threshold functions, and let h be their conjunction. We claim that h is consistent with $\mathbf{s}(G)$. Since $0 \geq \theta_l = -1/2$ it follows that $h_l(o) = 1$ for each l, and so $h(o) = 1$. In order to evaluate $h(v_i)$, note that if $\chi(i) = f$ then

$$\langle \alpha^{(f)}, v_i \rangle = \alpha_i^{(f)} = -1 < -1/2,$$

so $h_f(v_i) = 0$ and $h(v_i) = 0$, as required. Finally, for any colour l and edge ij we know that at least one of $\chi(i)$ and $\chi(j)$ is not l. Hence

$$\langle \alpha^{(l)}, v_i + v_j \rangle = \alpha_i^{(l)} + \alpha_j^{(l)},$$

where either both of the terms on the right-hand side are 1, or one is 1 and the other is -1. In any case the sum exceeds the threshold $-1/2$, and $h_l(v_i + v_j) = 1$. Thus $h(v_i + v_j) = 1$. □

The proof that the decision problem for consistency in P^k is NP-hard for $k \geq 3$ follows directly from this result. If we are given an instance G of *GRAPH k-COLOURING*, we can construct the training sample $\mathbf{s}(G)$ in polynomial time. If the consistency problem could be solved by a polynomial time algorithm A, then we could answer *GRAPH k-COLOURING* polynomial time by the following procedure: given G, construct $\mathbf{s}(G)$, and run A on this sample. The above result tells us that the answer given by A is the same as the answer to the original question. But *GRAPH k-COLOURING* is known to be NP-complete, and hence it follows that the P^k-*CONSISTENCY* problem is NP-hard if $k \geq 3$. (In fact, the same is true if $k = 2$. This follows from work of Blum and Rivest (1988).)

Thus, fixing k, we have a very simple family of feedforward linear threshold networks, each consisting of $k+1$ computation nodes (one of which is 'hard-wired' and acts simply as an AND gate) for which the problem of 'loading' a training sample is computationally intractable. Theorem 14 enables us to move from this hardness result for the consistency problem to a hardness result for pac learning. The theorem tells us that if we could pac learn P_n^k with running time polynomial in ϵ^{-1} and n then we could find a consistent hypothesis, using a randomised algorithm with running time polynomial in m and n. In the language of Complexity Theory this would mean that the latter problem is in RP, the class of problems which can be solved in 'randomised polynomial time'. It is thought that RP does not contain any NP-hard problems — this is the 'RP $\neq$ NP' conjecture, which is considered to be as reasonable as the 'P $\neq$ NP' conjecture. Accepting this, it follows that there is no polynomial time pac learning algorithm for the graded space $P^k = \bigcup P_n^k$ when $k \geq 2$.

This may be regarded as a rather pessimistic note, but it should be emphasised that the 'non-learnability' result discussed above is a worst-case result and indicates that training feedforward linear threshold networks is hard *in general*. This does not mean that a particular learning problem cannot be solved in practice.

8. EXTENSIONS AND GENERALISATIONS

The basic pac model is useful, but it has clear limitations. A number of extensions to the basic model have been made in the last few years. In this section, we briefly describe some of these. It is not possible to give all the details here; the reader is referred to the references cited for more information.

8.1 Stochastic concepts

The results presented so far have nothing to say if there is some form of 'noise' present during the learning procedure. Further, the basic model applies only to the learning of functions: each example is either a positive example or a negative example of the given target concept, not both. But one can envisage situations in which the 'teacher' has difficulty classifying some examples, so that the labelled examples presented to the 'learner' are not labelled by a function, the same example being on occasion presented by the 'teacher' as a positive example and on other occasions (possibly within the same training sample) as a negative example. For example, in the context of machine vision, if the concept is a geometrical figure then points close to the boundary of the figure may be difficult for the teacher to classify, sometimed being classified as positive and sometimes as negative. Alternatively, the problem may not lie with the teacher, but with the 'concept' itself. This may be ill-formed and may not be a function at all.

To deal with these situations, we have the notion of a *stochastic concept*, introduced by Blumer *et al.* (1989). A stochastic concept on X is simply a probability distribution P on $X \times \{0,1\}$. Informally, for finite or countable X, one interprets $P((x,b))$ to be the probability that x will be given classification b. This can be specialised to give the standard pac model, as follows. Supppose we have a probability distribution μ on X and a target concept t; then (see Anthony and Shawe-Taylor (1990), for example) there is a probability distribution P on $X \times \{0,1\}$ such that for all measurable subsets S of X,

$$P\left(\{(x,t(x)) \mid x \in S\}\right) = \mu(S); \quad P\left(\{(x,b) \mid x \in S, b \neq t(x)\}\right) = 0.$$

In this case, we say that P *corresponds* to t and μ. What can be said about 'learning' a stochastic concept by means of a hypothesis space H of $\{0,1\}$-valued functions? The error of $h \in H$ with respect to the target stochastic concept is the probability

$$\mathrm{er}_P(h) = P\left(\{(x,y) \in X \times \{0,1\} \mid h(x) \neq y\}\right)$$

of misclassification by h of a further randomly drawn training example. If P is truly stochastic (and not merely the stochastic representation of a function, as described above) it is unlikely that this error can be made arbitrarily small. As earlier, *observed error* of h on a training sample $\mathbf{s} = ((x_1,b_1),(x_2,b_2),\ldots,(x_m,b_m))$ is defined to be

$$\mathrm{er}_{\mathbf{s}}(h) = \frac{1}{m}\left|\{i \mid h(x_i) \neq b_i\}\right|.$$

Clearly this may be non-zero for all h (particularly if the same example occurs twice in the sample, but with different labels). What should 'learning' mean in this context? What we should like is that there is some sample size m_0, independent of the stochastic concept P, such that if a hypothesis has 'small' observed error with respect to a random sample of length at least m_0 then, with high probability, it has 'small' error with respect to P. The following result follows from one of Vapnik (1982) and was first presented in the context of computational learning theory by Blumer *et al.* (1989). (The result presented here is a slight improvement due to Anthony and Shawe-Taylor (1990).)

Theorem 16 *Let H be a hypothesis space of $\{0,1\}$-valued functions defined on an input space X. Let P be any probability measure on $S = X \times \{0,1\}$ (that is, P is a stochastic concept on X), let $0 < \epsilon < 1$ and let $0 < \gamma \leq 1$. Then the P^m-probability that, for $\mathbf{s} \in S^m$, there is some hypothesis from H such that $\mathrm{er}_P(h) > \epsilon$ and $\mathrm{er}_{\mathbf{s}}(h) \leq (1-\gamma)\mathrm{er}_P(h)$ is at most*

$$4\,\Pi_H(2m)\exp\left(-\frac{1}{4}\gamma^2\epsilon m\right).$$

Furthermore, there is a constant $K > 0$ such that if H has finite VC dimension d, then there is $m_0 = m_0(\delta, \epsilon, \gamma)$ satisfying

$$m_0 \leq \frac{K}{\gamma^2\epsilon}\left(d\log\left(\frac{1}{\gamma\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right)$$

such that if $m > m_0$ then, for $\mathbf{s} \in S^m$,

$$\mathrm{er}_{\mathbf{s}}(h) \leq (1-\gamma)\epsilon \Longrightarrow \mathrm{er}_P(h) < \epsilon$$

with probability at least $1 - \delta$. □

The notion of a stochastic concept can be applied to a number of situations. As already indicated, it can be useful when the target 'concept' is not a function. It can also be useful when there is 'classification noise' (see Angluin and Laird (1988)), that is, where there is an underlying target function, but the randomly chosen examples have their labels 'flipped' occasionally. This corrupts the training data and results in a stochastic concept. Additionally, in the standard pac model, we have assumed that the concept space is a subset of the hypothesis space. Suppose this is not so and $t \in C \setminus H$. Then there can be no $h \in H$ such that the error of h with respect to t is 0 for all probability distributions μ on X. However, Theorem 16 is applicable. Suppose μ is a distribution on X and take P to be the stochastic concept corresponding to t and μ. Since the sample size given in Theorem 16 is independent of the stochastic concept P, we obtain a type of learnability result when H has finite VC dimension: there is a sample size $m_0(\delta, \epsilon)$ such that if a randomly drawn training sample of t of length m_0 is presented, then with probability at least $1 - \delta$, if $h \in H$ is correct on a fraction of at least $1 - \epsilon/2$ of the sample, then h has error at most ϵ. (Here, we have taken $\gamma = 1/2$ for simplicity.) In fact, somewhat more can be said about 'learning' in this case. Suppose we have a concept space C and a hypothesis space H defined on X, not necessarily such that $C \subseteq H$. Suppose that μ is a probability distribution on X. For any target $t \in C$, let

$$\mathrm{opt}_H(t) = \inf_{h \in H} \mathrm{er}_\mu(h, t).$$

Let us say that a learning algorithm L for (C, H) is a *probably approximately optimal* algorithm if for any $0 < \delta, \epsilon < 1$, there is $m_0(\delta, \epsilon)$ such that for any $t \in C$ and any μ, if a training sample $\mathbf{s}$ for t of length at least m_0 is randomly drawn then with probability at least $1 - \delta$,

$$\mathrm{er}_\mu(L(\mathbf{s})) < \mathrm{opt}_H(t) + \epsilon.$$

We say that a hypothesis space H has the *uniform convergence of errors* (*UCE*) property if the following holds. Given real numbers δ and ϵ ($0 < \delta, \epsilon < 1$), there is a positive integer $m_0(\delta, \epsilon)$ such that, for any probability distribution P on $S = X \times \{0,1\}$,

$$P^m \left(\{\mathbf{s} \in S^m \mid \text{ for all } h \in H,\ |\text{er}_P(h) - \text{er}_{\mathbf{s}}(h)| < \epsilon\}\right) > 1 - \delta.$$

This means, roughly speaking, that one can guarantee with high confidence that the observed errors of functions in H are within ϵ of their actual errors. Results of Vapnik and Chervonenkis (1971) on the uniform convergence of relative frequencies to probabilities show that if H has finite VC dimension then H has the UCE property. It follows, upon taking P to be the stochastic concept corresponding to t and μ, that there is a probably approximately optimal learning algorithm for (C, H): the algorithm which returns the hypothesis which minimises the observed error. Of course, this minimisation may be a computationally intractable problem, but the emphasis here is on whether such learning is theoretically possible.

8.2 Variations on the pac model

We have observed that pac learning may be computationally intractable, due to the difficulty of the associated consistency problem. Many researchers have looked at a number of ways of varying the model to make learning easier. We shall now briefly discuss two important variations: distribution-dependent models and models which allow queries.

Perhaps the main attraction of the definition of pac learning is the 'distribution-free' criterion: the sample size is independent of the probability distribution. The proofs of the standard computational hardness results for pac learning and the lower bounds on sample complexity involve the use of very particular probability distributions, so the theory presented earlier is very dependent on this criterion. If we know in advance what the distribution on the example space is, or if we know that it is one of a particular set of distributions, then the full stength of the pac definition is not needed. Generally, suppose C is a concept space, H is a hypothesis space, and $\mathcal{P}$ is a class of probability distributions on X. We may say that a learning algorithm L for (C, H) is a $(C, H, \mathcal{P})$ pac learning algorithm if there is a sample size $m_0(\delta, \epsilon)$ such that for any $t \in C$ and *any* $\mu \in \mathcal{P}$, with μ^m-probability at least $1 - \delta$, if a training sample $\mathbf{s}$ is presented, $\text{er}_\mu(L(\mathbf{s})) < \epsilon$. This is a weakening of the pac criterion, in that m_0 need be uniform only over $\mathcal{P}$ and not over *all* distributions. The case $\mathcal{P} = \{\mu\}$, in which $\mathcal{P}$ consists of just one distribution, has been studied by Benedek and Itai (1988,1991), where a characterisation for learnability in terms of *ϵ-covers* is obtained. In general, finite VC dimension is not necessary for $(C, H, \mathcal{P})$ pac learning. (The theory of previous sections shows that it is necessary when $\mathcal{P}$ consists of all distributions.) In addition, learning with respect to a particular distribution may be computationally feasible in situations where standard pac learning is NP-hard. For further discussion of distribution-dependent learning, we refer the reader to the papers of Benedek and Itai (1992), Ben-David, Benedek and Mansour (1989), Bertoni *et al.* (1992), Kharitonov (1993), Li and Vitanyi (1989), Linial, Mansour and Nisan (1989).

In the standard pac framework, the learning algorithm receives labelled examples and forms a hypothesis only on the basis of these. The learning algorithm has no control over the sequence of training examples. Clearly, it might be possible to make learning easier if we allow L to 'ask questions', such as: 'is x a positive example?', for a particular x, chosen by the algorithm. This type of query is known as a membership query and, intuitively, one might expect that it makes it easier for the learning algorithm to converge to the target concept. The idea of learning with this and other types of query in addition to random labelled examples (and sometimes in place of random labelled examples) was mentioned by Valiant (1984a) and has been studied extensively in recent years. We refer the reader to the papers by Angluin (1988), Angluin, Frazier and Pitt (1990), Baum (1990,1991), Maass and Turan (1990,1992), and the survey by Angluin (1992) and the references therein.

8.3 The graph dimension

As far as applications to artificial neural networks are concerned, the most significant and important extension of the basic pac model is to the learning of general function spaces. The basic model concerns $\{0,1\}$-valued functions only; that is, it is concerned only with classification problems. We have seen how it applies to feedforward linear threshold networks having one output node. But what can be said about learning and generalisation in feedforward linear threshold networks having more than one output node, or in artificial neural networks with sigmoid activation functions and a real-valued output? To deal with these problems and others, the pac model has been extended in a number of ways. Most of the relevant work has been on sample complexity rather than computational complexity, with a few notable exceptions, such as the paper of Kearns and Schapire (1990). Here, we concentrate on the problem of sufficient sample size. In what follows, certain technical measure-theoretic conditions have to be satisfied; we shall not discuss these, but refer the reader to the paper of Haussler (1992) or to the book by Pollard (1984).

Suppose that C, H are sets of functions from an example space X into a set Y, with $C \subseteq H$, and suppose that $t \in C$. Suppose also that there is a probability distribution μ on X. Generalising in the obvious way from our previous definitions, we may define the *error* of $h \in H$ with respect to t to be

$$\mathrm{er}_\mu(h) = \mu(\{x \in X \mid h(x) \neq t(x)\}).$$

That is, h is erroneous on example x if $h(x) \neq t(x)$. When $Y = \mathbf{R}$, for example, this may seem a little coarse; we shall later discuss an alternative approach. With this measure of error, we may define learning as earlier.

For $h \in H$, let $\mathcal{G}h$ be the function from $X \times Y$ to $\{0,1\}$ defined by

$$\mathcal{G}h(x,y) = 1 \Longleftrightarrow h(x) = y.$$

and let $\mathcal{G}H = \{\mathcal{G}h : h \in H\}$. Now, it can be shown that if the hypothesis space $\mathcal{G}H$ is pac learnable (in the usual sense), then so too is H, by any consistent learning algorithm; see Natarajan (1989) and Anthony and Shawe-Taylor (1990). Furthermore, the sample complexity of any consistent learning algorithm for H (when regarded as a pac algorithm) can be bounded by an expression involving the VC dimension of $\mathcal{G}H$. (A suitable upper bound is any upper bound on the sample complexity of a consistent learning algorithm for $\mathcal{G}H$, such as the expression of Theorem 8 with $d = \mathrm{VCdim}(\mathcal{G}H)$.) This quantity $\mathrm{VCdim}(\mathcal{G}H)$ is known as the *graph dimension* of H (Natarajan (1989)) and is denoted $\mathrm{Gdim}(H)$. Clearly, when $Y = \{0, 1\}$, the graph dimension and the VC dimension coincide.

We remark that the idea of stochastic concept can be extended to 'stochastic function'. Indeed, the distribution P discussed above is an example of a stochastic function. More generally, the analysis presented earlier for (standard) stochastic concepts extends to stochastic functions; see Anthony and Shawe-Taylor (1990) and Buescher and Kumar (1992), for example, and later in this section, where a more general framework is described.

We see from this discussion that if $\mathrm{Gdim}(H)$ is finite then H is pac learnable by any consistent algorithm. It is natural to ask whether finite graph dimension is a necessary condition for learnability in this generalised model. However, Natarajan showed this not to be the case: there are pac learnable function spaces with infinite graph dimension (see Natarajan (1991)). Natarajan finds a weaker necessary condition for learnability, showing a certain measure, now known as the Natarajan dimension, must be finite for H to be pac learnable. More recently, Ben-David, Cesa-Bianchi and Long (1992) have shown that when Y is *finite*, the finiteness of the graph dimension is a necessary and sufficient condition for H to be pac learnable. Furthermore, they show that the Natarajan dimension is finite if and only if the graph dimension is finite, so that Natarajan's necessary and sufficient conditions are matching. In fact, they obtain a 'meta-result', characterising those measures of dimension which themselves characterise learnability in the case of finite Y.

The graph dimension has been applied to obtain bounds on the required sample size for learning in artificial neural networks. Natarajan (1989) obtained a result bounding the graph dimension of a linear threshold network (not necessarily a feedforward one) with any number of output nodes and with $\{0, 1\}$-valued inputs. For feedforward linear threshold networks with real inputs, Shawe-Taylor and Anthony (1991) (see also Anthony and Shawe-Taylor (1990)) generalised the result of Baum and Haussler (1989) presented in Theorem 12. Specifically, they showed that if a feedforward linear threshold network has any number k of output nodes then the graph dimension of the space of $\{0, 1\}^k$-valued functions it computes is at most $2W \log(ez)$, where z, W are, as earlier, the number of computation nodes and the number of variable weights and thresholds. Thus, the same upper bound on sample size as presented in Theorem 13 holds for this more general case.

8.4 The pseudo-dimension

We have seen that the graph dimension can be used to measure the expressive power of a hypothesis space of functions, in somewhat the same way as the VC dimension is used for $\{0,1\}$-valued hypothesis spaces. But there are other such measures, as discussed by Haussler (1992) and Ben-David, Cesa-Bianchi and Long (1992), for example. In passing, we have already mentioned the Natarajan dimension. We now introduce a very useful dimension, known as the pseudo-dimension. This was introduced by Pollard (1984) and is defined whenever the set of functions maps into $Y \subseteq \mathbf{R}$. (More generally, it may be defined when Y is any totally ordered set, but this shall not concern us here.) Let H be a set of functions from X to $\mathbf{R}$. For any $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in X^m$, and for $h \in H$, let

$$h(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_m))$$

and let $H(\mathbf{x}) = \{h(\mathbf{x}) : h \in H\}$. We say that $\mathbf{x}$ is pseudo-shattered by H if some translate $\mathbf{r} + H(\mathbf{x})$ of $H(\mathbf{x})$ intersects all orthants of $\mathbf{R}^m$. In other words, $\mathbf{x}$ is pseudo-shattered by H if there are $r_1, r_2, \ldots, r_m \in \mathbf{R}$ such that for any $\mathbf{b} \in \{0,1\}^m$, there is $h_{\mathbf{b}} \in H$ with $h_{\mathbf{b}}(x_i) \geq r_i \iff b_i = 1$. The largest d such that some sample of length d is *pseudo-shattered* is the *pseudo-dimension* of H and is denoted by $\mathrm{Pdim}(H)$. (When this maximum does not exist, the pseudo-dimension is taken to be infinite.) When $Y = \{0,1\}$, the definition of pseudo-dimension reduces to the VC dimension. Furthermore, when H is a vector space of real functions, then the pseudo-dimension of H is precisely the vector-space dimension of H; see Haussler (1992).

8.5 A framework for learning function spaces

When considering a space H of functions from X to $\mathbf{R}^k$, it seems rather coarse to say that a hypothesis h is erroneous with respect to a target t on example x unless $h(x)$ and $t(x)$ are precisely equal. For example, with a neural network having k real-valued outputs, it is extremely demanding that each of the k outputs be *exactly* equal to that which the target function would compute. Up to now, this is the definition of error we have used. There are other ways of measuring error, if one is prepared to ask not *is the output correct?* but *is the output close?* in some sense. Haussler (1992) has developed a 'decision-theoretic' framework encompassing many ways of measuring error by means of *loss functions.* We shall describe this framework in a way which also subsumes the discussion on stochastic concepts.

First, we need some definitions. A *loss function* is, for our purposes, a non-negative bounded function $l : Y \times Y \to [0, M]$ (for some M). Informally, the loss $l(y, y')$ is a measure of how 'bad' the output y is, when the desired output is y'. An example of a loss function is the *discrete loss function*, defined by $l(y, y') = 1$ unless $y = y'$, in which case $l(y, y') = 0$. Another useful loss function in the L^1*-loss*, which is defined when $Y \subseteq \mathbf{R}^k$. This is given by

$$l(y, y') = \frac{1}{k} \sum_{i=1}^{k} |y_i - y'_i|.$$

In both of these examples, the loss function is actually a metric, but there is no need for this. For example, a loss function which is not a metric and which has been usefully applied by Kearns and Schapire (1990), is the L^2-*loss* or *quadratic loss*, defined on $\mathbf{R}^k$ by

$$l(y, y') = \frac{1}{k} \sum_{i=1}^{k} (y_i - y'_i)^2.$$

There are many other useful loss functions, such as the L^∞-*loss*, the *logistic loss* and the *cross-entropy loss.* In order to simplify our discussion here, we shall concentrate largely on the L^1-loss, which seems appropriate when considering artificial neural networks. The reader is referred to the influential paper of Haussler (1992) for far more detailed discussion of the general decision-theoretic approach and its applications. As in our discussion of stochastic concepts, we consider probability distributions P on $X \times Y$. Suppose that $l : Y \times Y \to [0, M]$ is a particular loss function. For $h \in H$, we define the *error* of h with respect to P (and l) to be

$$\mathrm{er}_{P,l}(h) = \mathbf{E}_P \left(l((h(x), y) \right) = \int_{X \times Y} l(h(x), y) dP(x, y),$$

the expected value of $l(h(x), y)$. When P is the stochastic concept corresponding to a target function t and a probability distribution μ on X, then this error is $\mathbf{E}_\mu((h(x), t(x))$, the average loss in using h to approximate to t. Note that if l is the discrete metric then this is simply the μ-probability that $h(x) \neq t(x)$, which is precisely the measure of error used in the standard pac learning definition.

Suppose that a sample $\mathbf{s} = ((x_1, y_1), \ldots, (x_m, y_m))$ of points from $X \times Y$ is given. The *observed error* (or *empirical loss*) of h on this sample is

$$\mathrm{er}_{\mathbf{s},l}(h) = \frac{1}{m} \sum_{i=1}^{m} l(h(x_i), y_i).$$

The aim of learning in this context is to find, on the basis of a 'large enough' sample $\mathbf{s}$, some $L(\mathbf{s}) \in H$ which has close to optimal error with respect to P; specifically, if $\delta, \epsilon > 0$ and if, as in our discussion of probably approximately optimal learning,

$$\mathrm{opt}_H(P) = \inf\{\mathrm{er}_{P,l}(h) : h \in H\},$$

then we should like to have

$$\mathrm{er}_{P,l}(L(\mathbf{s})) < \mathrm{opt}_H(P) + \epsilon,$$

with probability at least $1 - \delta$. As before, 'large enough' means at least $m_0(\delta, \epsilon)$, where this is independent of P. As for the standard pac model and the stochastic pac model described earlier, this can be guaranteed provided we have a 'uniform convergence of errors' property. Extending the earlier definition, we say that a hypothesis space H of functions from X to Y has the *uniform convergence of errors* (*UCE*) property if for

$0 < \delta, \epsilon < 1$, there is a positive integer $m_0(\delta, \epsilon)$ such that, for any probability distribution P on $X \times Y$,

$$P^m\left(\{\mathbf{s} \mid \text{for all } h \in H,\ |\mathrm{er}_{P,l}(h) - \mathrm{er}_{\mathbf{s},l}(h)| < \epsilon\}\right) > 1 - \delta.$$

If this is the case, then a learning algorithm which outputs a hypothesis minimising the observed error will be a probably approximately optimal learning algorithm; see Haussler (1992) for further discussion. We should note that minimisation of observed error is not necessarily the 'simplest' way in which to produce a near-optimal hypothesis. Buescher (1992) has obtained interesting results along these lines.

8.6 The capacity of a function space

An approach to ensuring that a space of functions has the UCE property, which is described in Haussler (1992) and which follows Dudley (1984), is to use the notion of the *capacity* of a function space. For simplicity, we shall focus here only on the cases in which Y is a bounded subset of some $\mathbf{R}^k$, $Y \subseteq [0, M]^k$, and we shall use the L^1-loss function, which from now on will be denoted simply by l. Observe that the loss function maps into $[0, M]$ in this case. We first need the notion of an *ϵ-cover* of a subset of a pseudo-metric space. A *pseudo-metric* ∂ on a set A is a function from $A \times A$ to $\mathbf{R}$ such that

$$\partial(a, b) = \partial(b, a) \geq 0, \quad \partial(a, a) = 0, \quad \partial(a, b) \leq \partial(a, c) + \partial(c, b)$$

for all $a, b, c \in A$. An *ϵ-cover* for a subset W of A is a subset S of A such that for every $w \in W$, there is some $s \in S$ such that $\partial(w, s) \leq \epsilon$. W is said to be *totally-bounded* if it has an ϵ-cover for all $\epsilon > 0$. When W is totally bounded, we denote by $N(\epsilon, W, \partial)$ the size of the smallest ϵ-cover for W. To apply this to learning theory, suppose that H maps X into $[0, M]^k$ and that μ is a probability distribution on X. Define the pseudo-metric ∂_μ on H by

$$\partial_\mu(f, g) = \mathbf{E}_\mu\left(l(f(x), g(x)\right).$$

We shall define the ϵ-capacity of H to be

$$\mathcal{C}_H(\epsilon) = \sup_\mu N(\epsilon, H, \partial_\mu),$$

where the supremum is taken over all probability distributions μ on X. If there is no finite ϵ-cover for some μ, or if the supremum does not exist, we say that the ϵ-capacity is infinite. The definition just given is not quite the same as the definition given by Haussler; here, we take a slightly more direct approach because we are not aiming for the full generality of Haussler's analysis. Results of Haussler (1992) and Pollard (1984) provide the following uniform bound on the rate of convergence of observed errors to actual errors.

Theorem 17 *With the notation of this section, if P is any probability distribution on $S = X \times Y$, then*

$$P^m\left(\{\mathbf{s} \mid \text{there is } h \in H \text{ with } |\mathrm{er}_{P,l}(h) - \mathrm{er}_{\mathbf{s},l}(h)| > \epsilon\}\right) < 4\,\mathcal{C}_H\left(\epsilon/16\right) e^{-\epsilon^2 m/64M^2}$$

for all $0 < \epsilon < 1$. □

When $k = 1$ and H maps into $[0, M]$, the capacity can be related to the pseudo-dimension of H. Haussler (1992) (see also Pollard (1984)) showed that if $d = \mathrm{Pdim}(H)$ then

$$\mathcal{C}_H(\epsilon) < 2\left(\frac{2eM}{\epsilon} \ln \frac{2eM}{\epsilon}\right)^d .$$

This, combined with the above result, shows that, in this case, H has the UCE property and that the sufficient sample size is of order

$$m_0(\delta, \epsilon) = \frac{K}{\epsilon^2}\left(\mathrm{Pdim}(H) \log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right).$$

Thus, if H is a space of real functions and $\mathrm{Pdim}(H)$ is finite, then the learning algorithm which outputs the hypothesis with minimum observed error is a probably approximately optimal learning algorithm with sample complexity $m_0(\delta, \epsilon)$. Thus, in a sense, for pac learning hypothesis spaces of real functions, the pseudo-dimension takes on a rôle analogous to that taken by the VC dimension for standard pac learning problems.

8.7 Applications to artificial neural networks

We now illustrate how these results have been applied to certain standard types of artificial neural network. We shall consider here the feedforward 'sigmoid' networks. In his paper, Haussler (1992) shows how the general framework and results can also be applied to radial basis function networks and networks composed of product units. Referring back to our definition of a feedforward network, we assumed at that point that each activation function was a linear threshold function. Suppose instead that each activation function f_r is a 'smooth' bounded monotone function. In particular, suppose that f_r takes values in a bounded interval $[\alpha, \beta]$ and that it is differentiable on $\mathbf{R}$ with bounded derivative, $|f_r'(x)| \leq B$ for all x. (We shall call such a function a *sigmoid*.) The standard example of such a function is

$$f(x) = \frac{1}{(1 + e^{\theta - x})},$$

where θ is known as the threshold, and is adjustable. This type of sigmoid function, which we shall call a *standard sigmoid*, takes values in $(0, 1)$ and has derivative bounded by $1/4$. By proving some 'composition' results on the capacity of function spaces and by making use of the pseudo-dimension and its relationship to capacity for real-valued function spaces, Haussler obtained bounds on the capacity of feedforward artificial neural networks with general sigmoid activation functions. It is not possible to provide all the details here; we refer the reader to his paper. Before stating the next result, we need a further definition. The *depth* of a particular computation node is the number of arcs in the longest directed path from an input node to the node. The depth of the network is the largest depth of any computation node in the network. We have the following special case of a result of Haussler (1992).

Theorem 18 *Suppose that (N, A) is a feedforward sigmoid network of depth d, with z computation nodes, n input nodes, any number of output nodes, and W adjustable weights and thresholds. Let Δ be the maximum in-degree of a computation node. Suppose that each activation function maps into the interval $[\alpha, \beta]$. Let H be the set of functions computable by the network on inputs from $[\alpha, \beta]^n$ when the variable weights are constrained to be at most V in absolute value. Then for $0 < \epsilon \leq \beta - \alpha$,*

$$\mathcal{C}_H(\epsilon) \leq \left(\frac{2e(\beta - \alpha)d(\Delta V B)^{d-1}}{\epsilon} \right)^{2W},$$

where B is a bound on the absolute values of the derivatives of the activation functions. Further, for fixed V, there is a constant K such that for any probability distribution P on $X \times \mathbf{R}^k$, the following holds: provided

$$m \geq \frac{K}{\epsilon^2} \left(W \log\left(\frac{B^d}{\epsilon} \right) + \log\left(\frac{1}{\delta} \right) \right),$$

then with P^m-probability at least $1 - \delta$, a sample $\mathbf{s}$ from $(X \times \mathbf{R}^k)^m$ satisfies

$$|\mathrm{er}_{\mathbf{s},l}(h) - \mathrm{er}_{P,l}(h)| < \epsilon$$

for all $h \in H$. Moreover, there is K_1 such that if

$$m \geq \frac{K_1}{\epsilon} \left(W \log\left(\frac{B^d}{\epsilon} \right) + \log\left(\frac{1}{\delta} \right) \right),$$

and $\mathrm{er}_{\mathbf{s},l}(h) = 0$ then, with probability at least $1 - \delta$, $\mathrm{er}_{P,l}(h) < \epsilon$. □

This result shows that the space of functions computed by a certain type of sigmoid network has the UCE property. It provides an upper bound on the order of sample size which should be used in order to be confident that the observed error is close to the actual error. In particular, therefore, it follows that a learning algorithm which minimises observed error is probably approximately optimal, with sample complexity bounded by the bounds in the theorem.

The presence of the bound B on the absolute values of the derivatives of the activation functions means that this theorem does not apply to linear threshold networks, where the activation functions are not differentiable. Nonetheless, the sample size bounds are similar to those obtained for linear threshold networks. Furthermore, in the theorem, there is assumed to be some uniform upper bound V on the maximum magnitude of the weights. Recently, Macintyre and Sontag (1993) have shown that if every activation function is the *standard* sigmoid function and if there is one output node, then such a bound is not necessary. They show that the set of functions computed by a standard sigmoid network with unrestricted weights (and on unrestricted real inputs) has finite pseudo-dimension.

We remark that the results presented concerning sigmoid neural networks are upper-bound results. One cannot easily give lower bounds on the sample size as for standard pac learning. One reason for this is that, although pac learnable function spaces from X to *finite* Y can be characterised (as in Ben-David, Cesa-Bianchi and Long (1992)), no matching necessary and sufficient conditions are known for the more general problem of pac learning when Y is infinite. In other words, it is an open problem to determine a single parameter which quantifies precisely the learning capabilities of a general function space.

REFERENCES

Angluin (1988): D. Angluin, Queries and concept learning. *Machine Learning*, 2(4): 319–342.

Angluin (1992): D. Angluin, Computational learning theory: survey and selected bibliography, in *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing.*

Angluin, Frazier and Pitt (1990): D. Angluin, M. Frazier and L. Pitt, Learning conjunctions of Horn clauses. In *Proceedings of the Thirtieth-First IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington DC. (See also: *Machine Learning*, 9 (2-3), 1992: 174–164.)

Angluin and Laird (1988): D. Angluin and P. Laird, Learning from noisy examples, *Machine Learning*, 2: 343–370.

Anthony and Biggs (1992): M. Anthony and N. Biggs, *Computational Learning Theory: an Introduction*, Cambridge University Press.

Anthony, Biggs and Shawe-Taylor (1990): M. Anthony, N. Biggs and J. Shawe-Taylor, The learnability of formal concepts. In *Proceedings of the Third Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA.

Anthony and Shawe-Taylor (1990): M. Anthony and J. Shawe-Taylor, A result of Vapnik with applications, Technical report CSD-TR-628, Royal Holloway and Bedford New College, University of London. To appear, *Discrete Applied Mathematics.*

Bartlett (1992): P.L. Bartlett, Lower bounds on the Vapnik-Chervonenkis Dimension of multi-layer threshold networks. Technical report IML92/3, Intelligent Machines Laboratory, Department of Electrical Engineering and Computer Engineering, University of Queensland, Qld 4072, Australia, September 1992.

Baum (1990): E.B. Baum, Polynomial time algorithms for learning neural nets. In *Proceedings of the Third Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA.

Baum (1991): E.B. Baum, Neural net algorithms that learn in polynomial time from examples and queries, *IEEE Transactions on Neural Networks*, 2: 5–19.

Baum and Haussler (1989): E.B. Baum and D. Haussler, What size net gives valid generalization? *Neural Computation*, 1: 151–160.

Ben-David, Benedek and Mansour (1989): S. Ben-David, G. Benedek and Y. Mansour, A parameterization scheme for classifying models of learnability. In *Proceedings of the Second Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA.

Ben-David, Cesa-Bianchi and Long (1992): S. Ben-David, N. Cesa-Bianchi and P. Long, Characterizations of learnability for classes of $\{0,\ldots,n\}$-valued functions. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York.

Benedek and Itai (1988): G. Benedek and A. Itai, Learnability by fixed distributions, In *Proceedings of the 1988 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA.

Benedek and Itai (1991): G. Benedek and A. Itai, Learnability with respect to fixed distributions, *Theoretical Computer Science* 86 (2): 377–389.

Benedek and Itai (1992): G. Benedek and A. Itai, Dominating distributions and learnability, In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York.

Bertoni *et al.* (1992): A. Bertoni, P. Campadelli, A. Morpurgo, S. Panizza, Polynomial uniform convergence and polynomial sample learnability. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York.

Billingsley (1986): P. Billingsley, *Probability and Measure*, Wiley, New York.

Blum and Rivest (1988): A. Blum and R.L. Rivest, Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA. (See also: *Neural Networks*, 5 (1), 1992: 117–127.)

Blumer *et al.* (1989): A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Learnability and the Vapnik-Chervonenkis Dimension. *Journal of the ACM*, 36(4): 929–965.

Buescher (1992): K.L. Buescher, Learning and smooth simultaneous estimation of errors based on empirical data (PhD thesis), Report UILU-ENG-92-2246, DC-144, Coordinated Science Laboratory, University of Ilinois at Urbana-Champaign.

Buescher and Kumar (1992): K.L. Buescher and P.R Kumar, Learning stochastic functions by smooth simultaneous estimation, In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York.

Cormen, Leiserson and Rivest (1990): T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms.* MIT Press, Cambridge, MA.

Dudley (1984): R.M. Dudley, A course on empirical processes. *Lecture Notes in Mathematics*, 1097: 2–142. Springer Verlag, New York.

Ehrenfeucht *et al.* (1989): A. Ehrenfeucht, D. Haussler, M. Kearns and L. Valiant, A general lower bound on the number of examples needed for learning. *Information and Computation*, 82 (3): 247–261.

Garey and Johnson (1979): M. Garey and D. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

Grunbaum (1967): B. Grunbaum, *Convex Polytopes.* John Wiley, London.

Haussler (1992): D. Haussler, Decision theoretic generalizations of the pac model for neural net and other learning applications, *Information and Computation*, 100: 78–150.

Haussler *et al.* (1988): D. Haussler, M. Kearns, N. Littlestone and M. Warmuth, Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA. (Also, *Information and Computation*, 95 (2), 1991: 129–161.)

Haussler and Welzl (1987): D. Haussler and E. Welzl, Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2: 127–151.

Judd (1988): J.S. Judd, Learning in neural networks. In *Proceedings of the 1988 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA.

Kearns and Schapire (1990): M. Kearns and R. Schapire, Efficient distribution-free learning of probabilistic concepts. In *Proceedings of the Thirty-First IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington DC.

Kharitonov (1993): M. Kharitonov, Cryptographic hardness of distribution specific learning. To appear, *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, 1993.*

Li and Vitanyi (1989): M. Li and P. Vitanyi, A theory of learning simple concepts under simple distributions and average case complexity for the universal distribution. In *Proceedings of the Thirtieth IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington DC.

Linial, Mansour and Nisan (1989): N. Linial, Y. Mansour and N. Nisan, Constant depth circuits, Fourier transforms, and learnability. In *Proceedings of the Thirtieth IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington DC.

Maass (1992): W. Maass. Bounds for the computational power and learning complexity of Analog Neural Nets. Manuscript, Institute for Theoretical Computer Science, Technische Universitaet Graz, Austria, 1992. To appear, *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, 1993.*

Maass and Turan (1990): W. Maass and G. Turan, On the complexity of learning from counterexamples and membership queries. In *Proceedings of the Thirty-First IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington DC.

Maass and Turan (1992): W. Maass and G. Turan, Lower bound methods and separation results for on-line learning models, *Machine Learning* 9 (2-3): 107–145.

Macintyre and Sontag (1993): A. Macintyre and E.D. Sontag, Finiteness results for sigmoidal "neural" networks (extended abstract). To appear, *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing.*

Minsky and Papert (1969): M. Minsky and S. Papert, *Perceptrons.* MIT Press, Cambridge, MA. (Expanded edition 1988.)

Natarajan (1989): B.K. Natarajan, On learning sets and functions. *Machine Learning*, 4: 67–97.

Natarajan (1991): B.K. Natarajan, *Machine Learning: A Theoretical Approach*, Morgan Kaufmann.

Pitt and Valiant (1988): L. Pitt and L.G. Valiant, Computational limitations on learning from examples. *Journal of the ACM*, 35 (4): 965–984.

Pollard (1984): D. Pollard, *Convergence of Stochastic Processes*, Springer-Verlag, New York.

Rosenblatt (1959): F. Rosenblatt, Two theorems of statistical separability in the perceptron. In *Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, November 1958. Vol. 1.* HM Stationery Office, London.

Sauer (1972): N. Sauer, On the density of families of sets, *Journal of Combinatorial Theory (A)*, 13: 145–147.

Shawe-Taylor and Anthony (1991): J. Shawe-Taylor and M. Anthony, Sample sizes for multiple output threshold networks, *Network* 2: 107–117.

Shawe-Taylor, Anthony and Biggs (1993): J. Shawe-Taylor, M. Anthony and N. Biggs, Bounding sample size with the Vapnik-Chervonenkis dimension. To appear, *Discrete Applied Mathematics*, Vol. 41.

Valiant (1984a): L.G. Valiant, A theory of the learnable. *Communications of the ACM*, 27 (11): 1134–1142.

Valiant (1984b): L.G. Valiant, Deductive learning. *Philosophical Transactions of the Royal Society of London* A, 312: 441–446.

Vapnik (1982): V.N. Vapnik, *Estimation of Dependences Based on Empirical Data.* Springer Verlag, New York.

Vapnik and Chervonenkis (1971): V.N. Vapnik and A.Ya. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16 (2), 264-280.

Vapnik and Chervonenkis (1981): V.N. Vapnik and A.Ya. Chervonenkis, Necessary and sufficient conditions for the uniform convergence of means to their expectations. *Theory of Probability and its Applications*, 26 (3), 532–553.

Wilf (1986): H.S. Wilf, *Algorithms and Complexity.* Prentice-Hall, New Jersey.