University of California
Santa Cruz

Data filtering and distribution modeling algorithms for machine learning

A dissertation submitted in partial satisfaction
of the requirements for the degree of
Doctor of Philosophy
in
Computer and Information Sciences
by
Yoav Freund
September 1993

The dissertation of Yoav Freund is approved:

Manfred K. Warmuth

David Haussler

David P. Helmbold

Dean of Graduate Studies and Research

Contents

Data filtering and distribution modeling algorithms for machine learning

Yoav Freund

ABSTRACT

This thesis is concerned with the analysis of algorithms for machine learning. The main focus is on the role of the distribution of the examples used for learning. Chapters 2 and 3 are concerned with algorithms for learning concepts from random examples. Briefly, the goal of the learner is to observe a set of labeled instances and generate a hypothesis that approximates the rule that maps the instances to their labels.

Chapter 2 describes and analyses an algorithm for improving the performance of a general concept learning algorithm by selecting those labeled instances that are most informative. This work is an improvement over previous work by Schapire. The analysis provides upper bounds on the time, space and number of examples that are required for concept learning. Chapter 3 is concerned with situations in which the learner can select, out of a stream of random instances, those for which it wants to know the label. We analyze an algorithm of Seung et. al. for selecting such instances, and prove that it is effective for the Perceptron concept class. Both Chapters 2 and 3 show situations in which a carefully selected exponentially small fraction of the random training examples are sufficient for learning.

Chapter 4 is concerned with learning distributions of binary vectors. Here we present a new distribution model that can represent combinations of correlation patterns. We describe two different algorithms for learning this distribution model from random examples, and provide experimental evidence that they are effective.

We conclude, in Chapter 5, with a brief discussion of the possible use of our algorithms in real world problems and compare them with classical approaches from pattern recognition.

Keywords: machine learning, computational learning theory, example selection, selective sampling, distribution modeling

Acknowledgments

First of all, I would like to thank Manfred Warmuth and David Haussler for all that they have taught me, and their for their help and support during my graduate studies. It was through working with them that I have learned how to do research, and how exciting and satisfying the process can be. I would especially like to thank Manfred for the many ideas that he provided along the way, and David for teaching me how to work with people from other disciplines and how to write papers. I would like to thank David Helmbold for his help in writing this thesis, especially Chapter 2.

Major parts of this work are products of collaborative efforts. The work in Chapter 2 has greatly benefited from several discussions with Rob Schapire. Major contributions to this chapter have also been made by Manfred Warmuth, David Haussler, Eli Shamir and Wolfgang Maass. Chapter 3 is joint work with Sebastian Seung, Eli Shamir and Tali Tishby, and Chapter 4 is joint work with David Haussler.

The experimental part of the work described in Chapter 4 was done in collaboration with KB Sriram, and I would like to thank him for all his hard work.

I would like to thank the other people with whom I had the opportunity to work during my graduate studies. Naoki Abe, Nicolo Cesa-Bianchi and Phil Long in Santa Cruz. Sebastian Seung, Eli Shamir, Tali Tishby, Dana Ron, Ronitt Rubinfeld and Yoram Singer in the Hebrew University in Jerusalem, and Mike Kearns, Rob Schapire and Linda Sellie in AT&T Bell Labs in Murray Hill. I would also like to thank Manfred Opper and Nick Littlestone for some interesting discussions.

Thanks to all my friends in Santa Cruz for the wonderful times that we had together.

Thanks to my girlfriend, Becky Hess, for her love and encouragement.

Finally, I am very grateful for the generous financial support of the Regents of the University of California (graduate fellowship), the Office of Naval Research (Contract number N0014-86-K-0454) and the US-Israel Binational Science Foundation (BSF) (Grant no. 90-00189/2).

1. Introduction

The objective of machine learning is to build machines that learn from their experience. There are many potential uses for such machines. One is predicting future events, such as a earthquakes or stock market prices. Another is building machines for identifying structure in complex data, such as sequences of proteins or DNA. Yet another use is building computers that are controlled using spoken or hand-written language. In this case learning is needed because voice patterns and writing styles vary so much between different people that it is close to impossible to write computer code that will anticipate every possible variety. By using machine learning techniques the computer can be trained to adapt to each individual user by exposing it to identified instances of speech or writing of the user.

Computers have been used for a long time to find models that fit empirical data using linear regression methods or to predict the future using simple parameterized predictors. Today, as computers are getting faster, the trend is towards using non-linear and hierarchical models to describe very complex high dimensional data. Methods that go under titles such as "neural networks" [Rumelhart and McClelland, 1986, Hertz *et al.*, 1991], "hidden Markov chains" [Rabiner and Juang, 1986], and "radial basis functions" [Poggio and Girosi, 1989] are becoming increasingly popular. Such algorithms are used for tasks ranging from controlling the arms of a robot to playing backgammon and from predicting the price of commodities to recognizing human faces. In general, a machine learning algorithm searches for a simple model that explains the past experience of the algorithm and then uses this model to predict future events. Various disciplines provide mathematical frameworks in which to analyze and compare such algorithms. Among these disciplines are pattern recognition [Duda and Hart, 1973a], estimation theory [Vapnik, 1982], the theory of stochastic modeling [Rissanen, 1986], the theory of inductive inference [Gold, 1967, Angluin and Smith, 1983] and computational learning theory [Valiant, 1984a].

Computational learning theory is a part of theoretical computer science. For this reason its natural emphasis is on learning questions that arise in computation theory, such as learning finite automata and Boolean formulas. Emphasis is also put on bounding the computational resources that are required for learning. The type of analysis often combines combinatorial arguments of the type usually found in computer science together with probabilistic arguments. However, the tendency is to minimize the number of probabilistic assumptions and use, as much as possible, worst case assumptions. These general tendencies set some of the work in computational learning theory apart from work done in other mathematical disciplines that analyze machine learning. Still, much of the work is closely related to work done in the other disciplines.

A large variety of mathematical frameworks are used in Computational Learning Theory, each of which emphasizes different aspects of learning. In this section we give a brief survey of some of the popular frameworks and indicate how the work presented in this thesis is related to these frameworks. After that survey, we give a brief introduction to each of the three main chapters of this thesis.

In general, one assumes the existence of some model that describes the world, or the task, with which the learner is faced. The goal of the learner is to discover that model by observing the behavior of the world. In the simplest and most studied type of learning the hidden model is a *concept*. Informally, a concept is a

rule that divides the world into positive and negative examples. For instance, the concept of "being blue" divides all objects into those that are blue and those that are not blue. The learning algorithm is presented with examples of blue and non-blue objects and and is required to deduce the general rule. More formally, we define the set of all possible objects as the *instance space* and define concepts as functions from the concept space to the labels "+" and "−". An instance, together with its label is called an example. The goal of concept learning is to generate a (description of) another function, called the *hypothesis*, which is close to the concept, using a set of examples. In general, we require that the learning algorithm observes just a small fraction of the instance space and that the learner can *generalize* the information provided by these examples to instances that have not been previously observed. It is clear that in order to do that the learner must have some prior knowledge about the set of possible (or likely) concepts. This knowledge is defined in terms of the *concept class*, which is the set of all a-priori possible concepts.

The most demanding learning goal is that of *exact identification*. Here the learner is required to exactly identify the model of the world and describe it in a concise form using a predefined language. For example, suppose the instance space is the plane and that the concept to be learned is the area bounded by a polygon. In an exact identification task, the learner might be required to generate an exact description of the polygon using a sequence of the coordinates of the polygon's corners. The goal of exact identification is often too hard. Moreover, in real-world scenarios a short and exact description of the behavior of the world might not even exist. A less demanding and more realistic goal is to generate an *approximate* hypothesis in an unrestricted language. The quality of such a hypothesis is measured by comparing between the labels it assigns to instances and the correct labels. The more these labels agree, the higher the quality of the hypothesis. For example, in the polygon learning problem described above, an approximate identification might consist of a polygon such that the area of the symmetric difference between the concept polygon and the hypothesis polygon is small.

One of the most popular mathematical frameworks of approximate concept learning is the model introduced by Valiant in [Valiant, 1984a], also called the *distribution free* or PAC (probably approximately correct) learning model. In this framework the labeled instances that are given to the learner as examples are chosen independently at random from the instance space. The distribution over the instance space is arbitrary, and not known to the learner. The accuracy of the hypothesis generated by the learner is defined as the probability that the hypothesis assigns a wrong label to a randomly chosen instance. Or, in other words, as the probability measure of the symmetric difference between the hypothesis and the concept. Note that this is a reasonable way of measuring the error, because the hypothesis is required to be accurate only on those parts of the instance space which are likely to be observed during learning. In chapter 2 we present an algorithm that can increase the accuracy of distribution free algorithms. This work is an improvement over previous work by Schapire [Schapire, 1990]. One of the main outcomes of the analysis of this algorithm are improved upper bounds on the number of examples and on the computational resources required for learning in Valiant's model.

Some learning tasks, such as the design of a character recognizer, are best formulated as the generation of a hypothesis. After the learning process is completed, the hypothesis is fixed, and can be encoded into fast computer hardware that is part of the character recognizer. Other tasks, such as the prediction of stock prices, are better described in terms of iterative trials. In this case the hypothesis of the learner is not

fixed, but rather changes after each example and then used for predicting the next example. The quality of learning is then measured by the rate at which the error in the predictions decreases as a function of the number of past trials, or, alternatively, by the total number of errors that the predictor makes over all trials. This type of learning is called *online* learning.

In the types of learning described so far, the learner is essentially passive, observing the behavior of its environment but not taking any active part in the the generation of examples. In many situations the learner can take a much more active role, asking questions and performing experiments on which to base its hypotheses. In computational learning theory this type of learning is called *query* learning. A wide variety of queries have been studied, two of the most popular ones are *membership queries*, where the learner asks if a particular instance is a positive instance of a hidden concept and is answered yes or no, and *equivalence queries*, where the learner asks whether a particular hypothesis is correct, and is either answered positively or is given an example on which prediction of the hypothesis is incorrect. For a survey of query learning see [Angluin, 1988b]. The goal of most of the algorithms for learning using queries is exact identification of the underlying model. A less studied goal is to use queries in the context of approximate identification to reduce the number of labeled instances that the learner needs for learning. Previous work by Eisenberg and Rivest has shown that, in the PAC learning model, no significant reduction of this type is possible for a natural set of concept classes. These results have been strengthened by Turan [Turán, 1993]. In chapter 3 we show that if we allow the learner access to random *unlabeled* instances, this problem can sometimes be alleviated. We show that *selecting* instances to be used as membership queries out of the random unlabeled instances has important advantages over *constructing* queries based on past information. In particular, we show that the number of labeled examples required for learning the Perceptron concept class[1] can be drastically reduced using a simple method for query selection proposed by Seung et. al. [Seung *et al.*, 1992].

In the frameworks described above, the learner's main source of information are the *labels* of the instances. The instances can usually be considered to encode a "state of nature", while the labels can be seen as given by a knowledgeable "teacher". This type of learning is referred to as *supervised* learning. An alternative framework is *unsupervised* learning, here the learner observes only unlabeled instances, and tries to model nature without reference to any "correct" labels. One possible goal of unsupervised learning is to generate an approximation of the unknown distribution of the instances. The quality of the hypothesis is then measured using one of the standard measures of difference between distributions, such as the Kullback-Leibler divergence. In chapter 4 we present a distribution model for binary vectors. We discuss the differences between this model and other models used in unsupervised learning algorithms, and argue why this model is relevant for some natural learning problems. We present and analyze some learning algorithms for the unsupervised learning of this model. Finally, we give experimental evidence illustrating the power of the model and of our learning algorithms.

The main part of this thesis is divided into three chapters, each of which is self contained and can be read separately from the rest. In the rest of the introduction we give a brief sketch of the main results in each chapter. Each chapter discusses a different problem and uses a different mathematical framework for

[1] The Perceptron concept class will be defined in Chapter 3.

its analysis. However, on a conceptual level, there are some common themes that appear in several places. The most important one is the central role that the distribution of the instances has in concept learning. The most common assumption about the distribution of instances that is used in computational learning theory is the *distribution-free* assumption. In this case the distribution is *arbitrary* and *unknown* to the learner. This can be seen as a choice that is made by an adversary whose goal is to make learning as hard as possible. Another popular assumption is the *distribution specific* assumption, in which the distribution is supposed to be known and fixed, irrespective of the hidden concept. Several results in this thesis suggest a model in which the distribution of the instances is assumed to be related to the hidden target. In this case learning can be easier than in the other cases, because the learner can gain important information both from the labels of the instances and from the distribution of the instances.

Some classical pattern recognition methods are based on estimating the distribution of the instances. Our approach is different in that it works on directly estimating a hidden deterministic target concept. This approach can have an advantage when a deterministic target concept is a good approximation of the actual relation between instances and their labels. We discuss these issues further in Chapter 5. There we compare our approach with the pattern recognition approach and discuss the technical and conceptual problems that need to be resolved in order to better understand the performance of our algorithms in real world scenarios.

1.1 Boosting by majority

In order to describe the results in Chapter 2 in some more detail, we need a somewhat more complete description of Valiant's distribution free learning model. We reiterate the basic definitions, and add some notation. The goal of learning is to generate a hypothesis, h, which approximates a hidden target concept c. The concept is a function from an instance space, X, to the labels "+" and "−", and is chosen (arbitrarily) from a concept class $\mathbf{C}$. The hypothesis is also a mapping from X to $\{+,-\}$. The hypothesis is not restricted to any particular form but must be efficiently computable. There is an arbitrary distribution, $\mathcal{D}$, over the instance space, the form of this distribution is not known to the learner. In order to learn, the learning algorithm receives a *sample* which is a set of m examples. Each example is an instance, together with its label according to c. The instances are chosen independently at random according to $\mathcal{D}$. After some computation, the learning algorithm outputs a description of the hypothesis h. The *error* of the hypothesis is defined as the probability that an instance, chosen at random according to $\mathcal{D}$, is labeled differently by h and by c. We require that the error is smaller than some predefined *accuracy* parameter $\epsilon > 0$. However, as there is always some small chance that the algorithm receives an unrepresentative sample, we allow the algorithm to fail with some small probability δ. Both ϵ and δ are given as input to the algorithm, and both can be set arbitrarily close to zero. Moreover, the resources required by the algorithm to achieve the desired performance should increase at a polynomial rate with respect to $1/\epsilon$ and $1/\delta$. These resources include the number of examples, m, The running time of the algorithm, and the amount of memory used by the algorithm. [2]

[2] In the more complete model of PAC learning, one defines parameters that measure the complexity of the concept class $\mathbf{C}$, and the dependence of the resources required by the algorithm on these parameters should also be polynomial.

An algorithm meeting these requirements is called a polynomial PAC learning algorithm for the concept class **C**. For many concept classes it is hard to find such algorithms, moreover, for an increasing number of concept classes there are proofs that no efficient algorithm exist (modulo some technical assumptions [Kearns and Valiant, 1989, Kharitonov, 1993]). A natural question is how the requirements of the PAC learning model can be weakened to make the problem easier. One possible weakening of the model, appropriately called *weak* PAC learning was suggested by Kearns and Valiant [Kearns and Valiant, 1989]. In this model one omits the requirement that ϵ and δ can be arbitrarily small. For example, one can ask for a learning algorithm that with a probability of 10% generates a hypothesis whose error is smaller than 25%. Such a requirement seems to be much less demanding than that of the standard, or strong, PAC learning model. However, a surprising result by Schapire [Schapire, 1990], shows that the two models are essentially equivalent. He shows that if there exists a learning algorithm that can, with non-zero probability, generate a hypotheses whose error is smaller than $50\% - \gamma$, for some constant $\gamma > 0$, then this algorithm can be transformed into a strong PAC learning algorithm, that achieves arbitrary accuracy with arbitrarily high probability. Note that the hypothesis that randomly labels each example it sees using an unbiased coin flip always has an error of 50%, thus the requirement on the weak learning algorithm is essentially that its predictions are just slightly better than random guessing. The transformation presented by Schapire is given in terms of a general purpose "boosting" algorithm. This algorithm uses the weak learning algorithm as a procedure and combines several of the weak hypotheses generated by it into a single very accurate hypothesis. In addition to the surprising theoretical aspect of Schapire's result, his algorithm is very attractive for practical purposes, because it suggests a method for improving the performance of any learning algorithm. In Chapter 2 we present a different boosting algorithm. Our boosting algorithm achieves the same improvement in the performance of the weak learning algorithm as Schapire's algorithm using fewer runs of the weak learning algorithm and combining the weak hypotheses in a simpler way. Also, our algorithm requires less computational resources than Schapire's algorithm. In Chapter 2 we show that the number of times that the weak learner is called is optimal, and that the amount of computational resources is close to optimal. We shall now sketch the main ideas that are used in our boosting algorithm.

Consider a learning algorithm that always generates a randomized hypothesis such that the probability that the hypothesis is mistaken on any particular instance is $1/2 - \gamma$, independent of anything else. This is a weak PAC learning algorithm which is very easy to boost. One can simply run it n times, generating n different hypotheses, and then output the hypothesis that is the majority vote over the outputs of these hypotheses. The probability that the majority vote is incorrect on any particular instance is equal to the probability that a biased coin, whose probability for head is $1/2 + \gamma$, gets less than $n/2$ heads in n tosses. This probability decreases very rapidly as n increases, and thus the expected error of the majority hypothesis is very small. Of course, this is a very special type of a weak learner. A general weak learner, without any constraints, might generate the exact same hypothesis every time we call it, and then the majority hypothesis will be equal to each of those weak hypothesis and its error will still be $1/2 - \gamma$. A way in which we can prevent the weak learner from generating the same hypothesis over and over again is by exposing it to examples drawn according to different distributions over the instance space. For example, we can present the learning algorithm only with those examples on which the previous hypothesis is incorrect. In this case the hypothesis that the weak learner generates must (with high probability) be correct with

probability $1/2 + \gamma$ on those instances on which the previous hypothesis was incorrect, because this is exactly the part of the instance space with respect to which the quality of the hypothesis is measured. It might seem that in this way we can insure some progress towards a set of hypotheses whose errors can be reduced by a majority vote. However, note that the new hypothesis might be incorrect on *all* of the instances on which the previous hypothesis was correct. Thus while we are making some progress on one part of the instance space, we are losing all of the advantage that we had on another part. Luckily, there exists a way of generating distributions over the instance space that guarantee that progress is being made by each hypothesis generated by the learner, such that combining all the hypotheses by a majority vote generates an accurate hypothesis. Somewhat surprisingly, the *number* of hypotheses that needs to be combined is equal to the number that is required when the hypotheses are generated by the very simple "independent" weak learner described earlier. The basic idea of our boosting algorithm is to run the weak learning algorithm n times, each time exposing it to a distribution that is more concentrated (in a particular way) on those instances on which previous hypotheses have been incorrect. These n hypotheses are then combined into a single hypothesis by taking a majority vote over them.

One of the ways in which the boosting algorithm can generate the distributions to which it exposes the weak learner is by a method of *filtering*. This method was first used by Schapire in his boosting algorithm. Filtering is a process by which each example that is presented to the boosting algorithm undergoes a stochastic test. If the example passes the test, then it is *accepted* and passed on to the weak learner, otherwise it is discarded. An interesting fact is that almost all of the examples that are given to the boosting algorithm are discarded, and never take part in the actual learning process performed by the weak learner. More precisely, in order to generate a hypothesis whose error is smaller than ϵ, the algorithm tests $\tilde{O}(1/\epsilon)$ examples[3], out of which only $O(\log 1/\epsilon)$ examples are accepted. This might seem to be a very inefficient way of using the random examples. However, the total number of examples used for learning is very close to the lower bound on the number of examples needed for learning proven by Blumer et. al. [Blumer *et al.*, 1989]. In fact, the boosting algorithm makes such efficient use of its resources, that its analysis gives the best general upper bounds that are currently known on the resources required for PAC learning algorithms that use polynomial resources.

There are two interesting corollarys from the observation that most of the examples that are given to the boosting algorithm are discarded. The first is that most of the running time of the boosting algorithm is spent in the search for the small number of important examples and *not* in the weak learning algorithm that is generating the weak hypotheses. As the search for the important examples can be performed in parallel, we find that any PAC learning algorithm can be transformed to a form that can execute very efficiently on a parallel computer. Essentially, given some technical assumptions, if a parallel computer with a sufficient number of processors is available, any PAC learning algorithm can be run in time $O(\log 1/\epsilon)$. The second corollary is that if a concept class is learnable then the labels of *any* sample of size m, for large enough m, can be deduced from the labels of just $O(\log m)$ of the instances. However, note that in order to find this set of examples, the boosting algorithm needs to know the labels of all the instances in the sample. In Chapter 3, described in the next section, we show that in certain cases this small set of

[3]The notation $\tilde{O}(\cdot)$ is used to indicate that log factors are ignored.

important instances can be detected even without knowing the labels of the whole sample in advance.

It is clear that our boosting algorithm makes intensive use of the fact that the weak learning algorithm, in Valiant's model of learning, is required to generate an accurate hypothesis for *any* input distribution. This gives our algorithm its power, but also points to a potential problem of using it in practice. The problem is that most real world learning algorithms are not distribution independent, on the contrary, their performance is highly dependent on the distribution of instances with which they are presented. A natural question is whether one can boost the performance of such distribution-dependent learning algorithms. We give an affirmative answer to this question in Section 2.4.1. Our analysis shows that our boosting algorithm can be used with such algorithms, and that the accuracy of the hypothesis that it outputs is proportional to the sensitivity of the given learning algorithm to changes in the distribution of the instances.

Parts of this work were previously published in [Freund, 1990] and [Freund, 1992].

1.2 Query By Committee

As we have discussed in the previous section, all random training examples are not created equal. In fact, there is often a very small fraction of the training examples whose labels carry all the information that is relevant for approximating the hidden concept, and knowing these labels makes all the other labels redundant.

An interesting question is whether learning algorithms that have access to membership queries, i.e. algorithms that can ask for the label of any particular instance, can reduce the number of training examples by querying only on this small set of instances whose labels are the most informative.

This question was previously studied by Eisenberg and Rivest [Eisenberg and Rivest, 1990] in the PAC learning framework. They give a negative result, and show that for a natural set of concept classes, which they call "dense in themselves", queries can not decrease the number of labels that the learner has to observe before it can generate an accurate hypothesis. Intuitively, the reason is that some of the information that is conveyed to the learner by a random sample can not be gathered from queries. When a leaner is given a sample of random examples, it not only receives information about the target concept, but it also gets an empirical estimate of the distribution of the instances. This estimate is important because the error of the hypothesis is measured with respect to the distribution of the instances. Clearly, membership queries that are constructed by the learning algorithm convey no information about the distribution of the instances.

In Chapter 3 we present a framework of learning from queries in which this problem can be overcome. In this framework, the learner is given separate access to the random instances and to their labels. We assume that receiving a random *unlabeled* instance is relatively cheap, while obtaining the label of a random instance is more expensive. In many real life cases, such as character or speech recognition, this is a natural assumption, because gathering random unlabeled examples is a basically automatic process, while obtaining the correct label of an example requires human labor.

We study one particular algorithm for learning in this framework, that was presented by Seung et. al. [Seung *et al.*, 1992], called "Query-by-Committee". The algorithm uses a "committee" of learners, that is to say, a set of independent learning algorithms, each of which generates a hypothesis that is consistent

with the answers to all the queries asked so far. The algorithm selects its membership queries from among the random instances. It asks membership queries on those random instances that cause disagreement among the committee members, in other words, those examples that are labeled differently by different consistent hypotheses.

The process of learning can be seen as an interplay between two types of information. On the one hand, labeled instances convey information about the hidden concept, this information can be measured by the number of hypotheses that are eliminated when the label of an instance is revealed. On the other hand, the set of hypotheses that is consistent with past experience can be used to predict the label of an unlabeled instance. Our analysis of the query by committee algorithm is based on the analysis of this interaction. We show that for the Perceptron concept class, the chosen membership queries reduces the set of consistent hypothesis at a fast rate. We also show that, in general, when such fast reduction is achieved by the Query by Committee algorithm, and the concept class is learnable, then the prediction error decreases exponentially fast in the number of queries asked.

Part of this work was previously published in [Freund *et al.*, 1993].

1.3 Learning distributions of binary vectors

The task of the learner in Chapters 2 and 3 is to generate a hypothesis that approximates a hidden concept, mapping instances to $\{+,-\}$. Tasks of this type are often referred to as *supervised* learning tasks. The reason for the name is that the instance labels can be assumed to be generated by a knowledgeable supervisor, or teacher. As we have discussed earlier, the distribution of the instances plays a major role in the process of learning, however, learning the form of this distribution is not the ultimate task of the learner. In contrast, in the framework of *unsupervised* learning, the goal of the learner is to generate a hypothesis based on unlabeled instances alone. The hypothesis is an approximate description of the distribution of the instances.[4] This description can sometimes provide important information about the process that generates the instances.

Consider, for example, a situation in which a learning algorithm is presented with statistical information regarding the people that visit a particular coffee shop and wishes to deduce a good distribution model for this information. For concreteness, suppose that the clientele of the coffee shop is a predefined set of n people, and that the learner is given a table that summarizes which of these people visited the coffee shop at each day of a particular year. For simplicity, we assume that the vector that defines the visitors on each day is generated by an independent random draw from a distribution that does not change over time. Clearly, one can deduce important information about social relationships from this table. Groups of people that tend to visit the coffee shop together are likely to be friends or take the same train, while people that tend not to be at the coffee shop at the same time might have some animosity towards each other. A good distribution model of the table should be able to capture and convey these friendships and animosities.

[4]Some unsupervised learning algorithms generate hypotheses that are not explicit distribution models. For example, some clustering algorithm generate a set of so-called "cluster centers". However, these cluster centers can usually be interpreted as the means of the components of a Gaussian (or related) mixture distribution.

In Chapter 3 we propose a class of simple distribution models over fixed length binary vectors that attempts to encode complicated distributions of this type in a useful way. The idea is to introduce a set of so-called "hidden variables" that represent events that cannot be directly observed, but have influence on the observable variables. Variables of this type are popular in so-called "Connectionist" models, such as Boltzmann Machines [Peterson and Anderson, 1987] and Belief Networks [Pearl, 1988]. These popular models can be interpreted as models of distributions over the observable variables. However, the distributions defined in this way cannot be described by a reasonably simple closed form expression. Our approach is to use a very restricted type of Boltzmann Machine which we call the "Combination Machine". This restriction makes the distribution model simple enough to enable us to describe it with a simple closed-form expression. However, it is still general enough to approximate an arbitrary distribution over the binary vectors to within any desired degree of accuracy.

We present two algorithms for the unsupervised learning of this model from random unlabeled instances. The first algorithm is a standard gradient ascent algorithm for maximizing the likelihood of the distribution model. The other model is an adaptation of the "Projection Pursuit" algorithm developed by Friedman and Tukey [Friedman and Tukey, 1974, Friedman *et al.*, 1984, Friedman, 1987], and by Huber [Huber, 1985]. The basic idea of this algorithm is that the important structure of a high dimensional distribution is conveyed by those projections of the distribution that are most different from the normal distribution. These algorithms are especially appealing because they allow an incremental construction of the distribution model and are thus much more efficient than the gradient ascent algorithm.

We have only a partial analysis of the learning algorithms that we present for the combination machine. However, we have performed some experiments that demonstrate the performance of these algorithms. Some of our experiments are on synthetic data which is generated using a combination machine. The goal of the learning algorithm in these experiments is to uncover the parameters of this machine from the distribution of the data. In other experiments we have used our algorithm to generate a model for the distribution of images of handwritten digits. In these experiments we show that the distribution model that is generated is meaningful and that it competes favorably with another popular distribution model.

Part of this work was previously published in [Freund and Haussler, 1992].

2. Boosting a weak learning algorithm by majority

2.1 Introduction

The field of computational learning is concerned with mathematical analysis of algorithms that learn from their experience. One of the main problems studied in computational learning theory is that of *concept learning*. Informally, a concept is a rule that divides the world into positive and negative examples. For instance, the concept of "being blue" divides all objects into those that are blue and those that are not blue. The learning algorithm is presented with examples of blue and non-blue objects and is required to deduce the general rule. More formally, we define the set of all possible objects as the *instance space* and define concepts as functions from the instance space to the labels "$-$" and "$+$". An instance, together with its label is called an example. The goal of concept learning is to generate a (description of) another function, called the *hypothesis*, which is close to the concept, using a set of examples. In general, we require that the learning algorithm observes just a small fraction of the instance space and that the learner can *generalize* the information provided by these examples to instances that have not been previously observed. It is clear that in order to do that the learner must have some prior knowledge about the set of possible (or likely) concepts. This knowledge is defined in terms of the *concept class*, which is the set of all a-priori possible concepts.

In this paper we study concept learning in a probabilistic setting. Here the examples that are given to the learning algorithm are generated by choosing the instances at random from a distribution over the instance space. This distribution is arbitrary and unknown to the learner. The central measure of the quality of a learning algorithm in the probabilistic setting is the accuracy of the hypotheses that it generates. The accuracy of a hypothesis is the probability that it classifies a random instance correctly. The accuracy of the hypotheses that are generated by a learning algorithm is expected to improve as the resources available to the algorithm are increased. The main resources we consider are the number of examples used for learning and the time and space available to the learning algorithm. One of the main results of this paper is an upper bound on the resources required for learning in the distribution-free model of learnability introduced by Valiant [Valiant, 1984a].

In Valiant's model, commonly referred to as the PAC (Probably Approximately Correct) learning model, or the *distribution-free* learning model, the quality of a learning algorithm is defined as follows. A learner is said to have *accuracy* ϵ with *reliability* $1-\delta$ if the probability, over the random choice of the examples and possible internal randomization of the learning algorithm, of generating a hypothesis that has error smaller than ϵ, is larger than $1-\delta$.[1]

As was recognized by Haussler et. al. [Haussler *et al.*, 1991a], increasing the reliability of any learning algorithm is easy. This can be done by testing the hypothesis generated by the algorithm on an independent set of examples to validate its accuracy. If the accuracy is not sufficient, the algorithm is run again, on a

[1] The exact definition of the PAC learning model are given in Section 2.3.1.

new set of random examples. It is easy to show that increasing the reliability from $1-\delta_1$ to $1-\delta_2$ can be achieved by running the algorithm $O(\log(1/\delta_2)/(1-\delta_1))$ times.[2]

Improving the accuracy of a learning algorithm is much harder. Two different variants of the PAC model were introduced by Kearns and Valiant [Kearns and Valiant, 1989] to address this issue. In *strong* PAC learning, which is the more common model, the learner is given the required accuracy, ϵ, as input, and is required to generate a hypothesis whose error is smaller than ϵ. The resources used by the algorithm can grow at most polynomially in $1/\epsilon$. On the other hand, in *weak* PAC learning the accuracy of the hypothesis is required to be just slightly better than $1/2$, which is the accuracy of a completely random guess. Kearns and Valiant proved that weak and strong learning are distinct for *distribution specific* learning. They have shown that while weak PAC learning of monotone Boolean functions with respect to the uniform distribution can be done in polynomial time, strong PAC learning of the same class will imply an ability to break some hard cryptographic problems that are commonly assumed to be unbreakable.

This seemed to indicate that weak and strong distribution-free learning should also be separated. However, Schapire [Schapire, 1990] proved that weak and strong PAC learning are equivalent in the distribution-free case. Schapire presented an algorithm that, given access to a weak learning algorithm, can generate hypotheses of arbitrary accuracy using time and space resources that are polynomial in $1/\epsilon$. This algorithm is called the "boosting" algorithm. The main idea is to run the weak learning algorithm several times, each time on a different distribution of instances, to generate several different hypotheses. We refer to these hypotheses as the "weak" hypotheses. These weak hypotheses are combined by the boosting algorithm into a single more complex and more accurate hypothesis. The different distributions are generated using an ingenious "filtering" process by which part of the random examples that are presented to the boosting algorithm are discarded, and only a subset of the examples are passed on to the weak learning algorithm. It turns out that corollaries of this important result give good upper bounds on the time and space complexity of distribution-free learning. Schapire's result also has many important implications related to group-learning, data-compression, and approximation of hard functions.

In this article we present a simpler and more efficient boosting algorithm. Schapire's boosting algorithm is defined recursively. Each level of the recursion is a learning algorithm whose performance is better than the performance of the recursion level below it. The final hypothesis it generates can be represented as a circuit consisting of many three-input majority gates. The input to the circuit are the labels produced by the weak hypotheses, and the output is the final label (see Figure 2.1). The depth of the circuit is a function of the problem parameters (accuracy and reliability), and its structure can vary between runs. The definition of our boosting algorithm, on the other hand, is not recursive and the final hypothesis can be represented as a single majority gate. This majority gate combines the outputs of all of the weak hypotheses.

In this paper we present two variants of our boosting algorithm. The first is *boosting by finding a consistent hypothesis*. This variant of the algorithm finds a hypothesis which is consistent with a large set of training examples. The analysis of this variant is quite straight forward, and its performance is close to the best performance we achieve. It also seems to be the variant whose application to practical learning

[2] A full analysis of this algorithm is given in Appendix A.1.

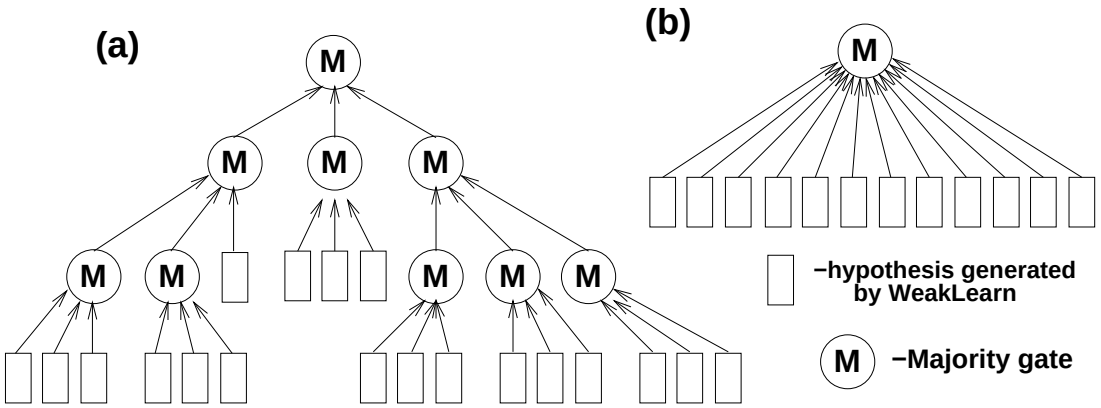


Figure 2.1: Final concepts structure: (a) Schapire (b) A one-layer majority circuit.

problems is more efficient [Drucker, 1992 1993]. The major drawback of this method is that it requires storage of the whole training set, which makes the space complexity dependence on ϵ be $O((\log 1/\epsilon)^2/\epsilon)$ (assuming that the concept class is fixed and that its VC dimension is finite). While this cost is often taken for granted, Schapire's algorithm demonstrates that boosting can be achieved using only $O(\log 1/\epsilon)$ space. We thus present a second variant of our algorithm, which we call *boosting by filtering*. This algorithm selects a small subset of the training examples as they are generated, and rejects all other examples. The sample complexity (number of training examples) of this version of the algorithm with respect to ϵ is $O((1/\epsilon)(\log 1/\epsilon)^{3/2}(\log\log 1/\epsilon))$, its time complexity is $O((1/\epsilon)(\log 1/\epsilon)^{5/2}(\log\log 1/\epsilon))$, its space complexity is $(\log 1/\epsilon)(\log\log 1/\epsilon))$ and the number of weak hypotheses it combines is $O(\log 1/\epsilon)$. These are, to the best of our knowledge, the best general upper bounds on the dependence of the resources required for computationally efficient PAC learning on the desired accuracy ϵ. We present some lower bounds that show that the possibilities for additional improvement are very limited. In particular, we show that there cannot be a general boosting algorithm that combines a smaller number of weak hypotheses to achieve the same final accuracy.

We also present generalizations of the algorithm to learning concepts whose output is not binary. One generalization is for concepts with k-valued outputs and is quite straightforward. Another generalization is to real-valued concepts. We show how boosting can be used in this case to transform a learning algorithm that generates functions whose expected error over the domain is bounded by c into a learning algorithm that generates functions whose error is bounded by $2c$ over most of the domain.

We also extend our result to distribution-specific learning. We show that our algorithm can be used for boosting the performance of learning algorithms whose quality depends on the distribution of the instances. More precisely, we show that the accuracy of the hypothesis that is generated by our boosting algorithm for a distribution $\mathcal{D}$ is a function of the rate in which the accuracy of the boosted learning algorithm decreases as the distribution of instances that it observes diverges from $\mathcal{D}$. The divergence between the distributions of the instances can be measured by the Kullback-Leibler measure of divergence.

Schapire [Schapire, 1992], noted that the results presented in this paper can be used to show an interesting relationship between representation and approximation using majority gates. These results

were independently discovered by Hastad et. al. [Goldmann *et al.*, 1992]. However, while their proof technique is very elegant, our proof is more constructive (for details see Section 2.2.2).

It is surprising to note that the boosting algorithm uses only a small fraction of the examples in the training set. While it needs $\Omega(1/\epsilon)$ examples to generate a hypothesis that has accuracy ϵ, only $O(\log 1/\epsilon)$ of them are passed to the weak learners. Two interesting implications arise from this fact. The first implication was pointed out to us by Schapire [Schapire, 1992]. It can be shown that if a concept class is learnable then the following type of compression can be achieved: Given a sample of size m, labeled according to some concept in the class, the boosting algorithm can be used to find a subsample of size $O(\log m)$ such that the labeling of all of the instances in the sample can be reconstructed from the labels of the subsample. The second implication was found jointly with Eli Shamir [Shamir, 1992]. We observed that if training examples can be accumulated in parallel by several parallel processors, then our methods can translate any PAC learning algorithm to a version that runs in time $O(\log 1/\epsilon)$ on a parallel computer with $\Theta(1/\epsilon)$ processors. This is because most of the examples that are given to the boosting algorithm are simply discarded and the search for a "good" example can be done by many processors in parallel.

The Paper is organized as follows. The main Theorem on which our boosting algorithms are based is given in Section 2.2 using a simple game-theoretic setting that avoids some of the complications of the learning problem while addressing the main underlying problem. In Section 2.3 we relate the theorem back to the learning problem, in Section 2.4 we present some extensions, and in Section 2.5 we summarize and present some open problems.

In Section 2.2 we present a game, called the "majority-vote" game, between two players, a "weightor" and a "chooser". The game consists of k iterations. For simplicity we now assume that the game is played on the set $\{1 \ldots N\}$. In each iteration the weightor assigns to the N points non-negative weights that sum to 1. The chooser has to then "mark" a subset of the points whose weights sum to at least $1/2 + \gamma$, where $0 < \gamma \leq 1/2$ is a fixed parameter of the game. The goal of the weightor is to force the chooser to mark each point in the space in a majority of the iterations, i.e. each point has to receive more than $k/2$ marks. We show that there exists a strategy that lets the weightor achieve that goal in $\lceil \frac{1}{2}\gamma^{-2} \ln N \rceil$ iterations. A similar game can be played on a general probability space, in which case the goal of the weightor is to force the chooser to mark all but and ϵ fraction of the space in the majority of the iterations. We show that $k = \lceil \frac{1}{2}\gamma^{-2} \ln 1/\epsilon \rceil$ iterations suffice in this case.

The weightor in this game represents the centerpiece of the boosting algorithm, which is the choice of the distributions that are presented to the weak learning algorithm. The points that the chooser decides to mark correspond to the instances on which the weak learner makes the correct prediction. This represents the freedom of the weak learner to distribute the error of the hypothesis in any way it chooses as long as the probability that a random instance is labeled correctly is at least $1/2 + \gamma$. This abstraction bypasses some of the complexities of the PAC learning problem, and can be read independently of the rest of the paper. In Subsection 2.2.1 we show that in the case of continuous probability spaces, there is a strategy for the chooser such that for any strategy of the weightor, if the game is stopped in less than $k = \lceil \frac{1}{2}\gamma^{-2} \ln 1/\epsilon \rceil$ iterations, more than ϵ of the space is marked less than $k/2$ times, i.e. the weightor fails to achieve its goal. Thus our weighting strategy is optimal for the case of continuous probability spaces. In Subsection 2.2.2

we present the implication of our analysis of the majority-vote game on the representational power of threshold circuits.

In Section 2.3 we relate the majority-vote game to the problem of boosting a weak learner and present the two variants of the boosting algorithm and their performance bounds. In order to simplify our analysis we restrict our analysis in Subsections 2.3.2 and 2.3.3 to the case in which the weak learning algorithms are deterministic algorithms that generate deterministic hypotheses. In Subsection 2.3.4 we show that this analysis needs to be changed only slightly to accommodate randomized learning algorithms that generate randomized hypotheses. In order to present the complete dependence of our bounds on the parameters of the problem, we don't use the notational conventions of polynomial PAC learning in our main presentation, but rather give explicit bounds including constants. Later, in Subsection 2.3.5, we derive upper bounds on the resources required for polynomial PAC learning that are the best general upper bounds of this type that exist to date. In Subsection 2.3.6 we compare our upper bounds to known lower bounds and show that which aspects of our bounds are optimal and which might be further improved.

In Section 2.4 we give several extensions and implications of our main results. In Subsection 2.4.1 we show that our algorithm for boosting by filtering can work even in situations where the error of the hypotheses generated by the weak learning algorithm is not uniformly bounded for all distributions. In Subsection 2.4.2 we present a version of the boosting algorithm that works for concepts whose range is a finite set, and in Subsection 2.4.3 we present a version that works for concepts whose range is a real valued. In Subsection 2.4.4 we show how boosting can be used to parallelize learning algorithms. We conclude the paper with a summary and a list of open problems in Section 2.5. In the appendixes to the paper we give a summary of our notation and proofs of three lemmas.

2.2 The majority-vote game

In this section we define a two-player, complete information, zero-sum game. The players are the "weightor", D, and the "chooser", C. The game is played over a probability space $\langle X, \Sigma, V \rangle$, where X is the sample space, Σ is a σ-algebra over X, and V is a probability measure. We shall refer to the probability of a set $A \in \Sigma$ as the *value* of the set and denote it by $V(A)$. A real valued parameter $0 < \gamma \leq 1/2$ is fixed before the game starts.

The game proceeds in iterations, in each iteration:

1. The weightor picks a *weight* measure on X. The weight measure is a probability measure on $\langle X, \Sigma \rangle$. We denote the weight of a set A by $W(A)$.
2. The chooser selects a set $U \in \Sigma$ such that $W(U) \geq \frac{1}{2} + \gamma$, and *marks* the points of this set.

These two-step iterations are repeated until the weightor decides to stop. It then receives, as its payoff, the subset of X that includes those points of X that have been marked in more than half of the iterations played (if the number of iterations is even this set *does not* include points that have been marked exactly half the time). We shall refer to this set as the *reward* set and to its value as the *reward*. The complement of the reward set is the *loss* set. The goal of the weightor is to maximize the reward, and the goal of the chooser is to minimize it.

The question about this game in which we are interested is whether there exists a general strategy, independent of the specific probability space that guarantees the weightor a large reward. An affirmative answer to this question is given in this section. We describe a general strategy for the weightor such that for any probability space $\langle X, \Sigma, V \rangle$ and any $\epsilon, \gamma > 0$, the weightor can guarantee that the reward is larger than $1 - \epsilon$ after just $\frac{1}{2}(\frac{1}{\gamma})^2 \ln \frac{1}{\epsilon}$ iterations.

We shall present the weighting strategy in the following way. We start by giving some insight, and show what weighting strategies are reasonable. We then present the weighting strategy, and prove a bound on the reward that it guarantees. Finally we show that for non-singular sample spaces (such as a density distribution on R^n) there is a matching strategy for the adversary, implying that our strategy is the optimal minimax strategy when the sample space is non-singular.

In the following discussion we are fixing a particular instance of the game, i.e. we consider a particular sequence of moves taken by the two players. Let k be the number of iterations in the game. For $0 \leq i \leq k$ define $\{X_0^i, X_1^i, \ldots, X_i^i\}$ to be a partition of X into $i+1$ sets where X_r^i consists of those points in X that have been marked r times after i turns of the game have been played.

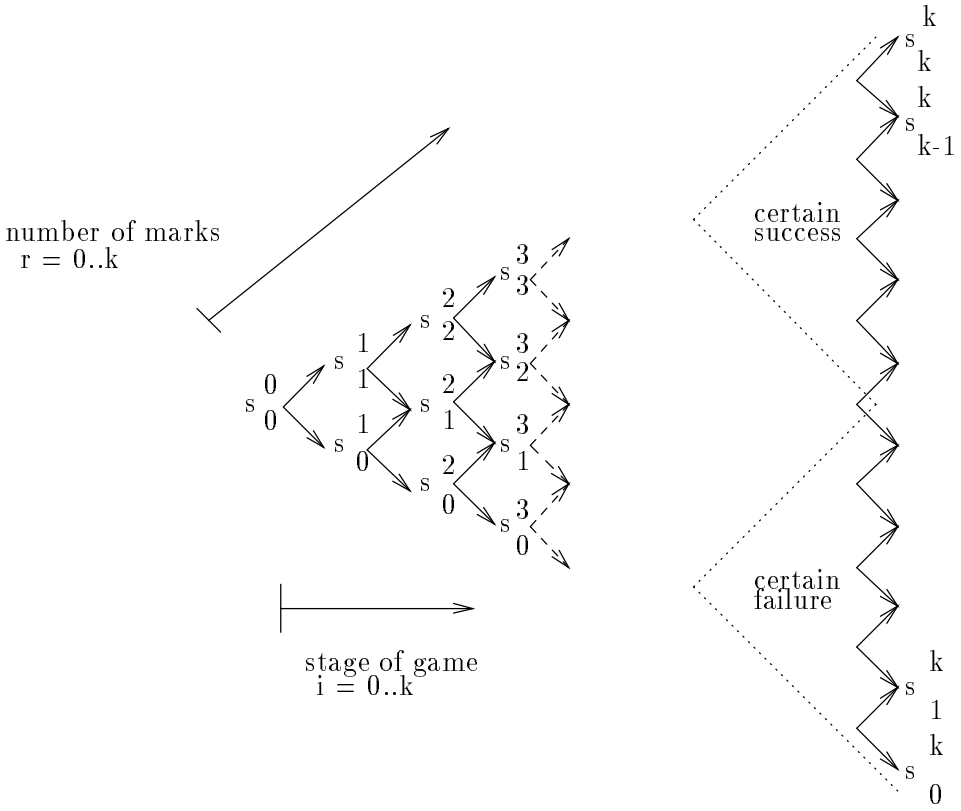


Figure 2.2: Transitions between consecutive partitions

As graphically presented in Figure 2.2, in iteration i, the chooser decides for each point in X_r^i whether to mark it or not, thus placing it in X_{r+1}^{i+1} or in X_r^{i+1}. The goal of the chooser is to minimize the value of the reward set: $\cup_{r=\lfloor k/2 \rfloor + 1}^{k} X_r^k$. The goal of the weightor is to maximize this value. By giving some points more weight than others, the weightor forces the chooser to mark more of those points. In the extreme,

by placing all the weight on a single point it guarantees that this point will be marked while at the same time allowing the chooser not to mark any other point, moving them closer to the loss set.

Let us define some notation:

k	the total number of iterations the game is played.
X_r^i	the set of points that have been marked r times in the first i iterations.
$M_r^i = X_r^i \cap X_{r+1}^{i+1}$	the subset of X_r^i that is marked in iteration i.
$q_r^i = V(X_r^i)$	the value of X_r^i.
$x_r^i = \frac{V(M_r^i)}{V(X_r^i)}$	the fraction of X_r^i that is marked in iteration i.
L	the loss set, i.e. those points that are in the end marked less than or equal to half the time.

Note that $X_0^0 = X$ and thus $q_0^0 = 1$.

Observe that if $r > k/2$, then points in X_r^i are guaranteed to be in the reward set. Likewise, if $i - r \geq k/2$ then points in X_r^i are guaranteed to be in the loss set. Thus it is intuitively clear that any reasonable weighting scheme will give zero weight to these points and place all the weight on those points for which both failure and success are still possible. In particular, the only points that should be assigned a non-zero weight in the final iteration are points in $X_{\lfloor k/2 \rfloor}^{k-1}$. We now present a weighting strategy that agrees with this intuition, and prove that this strategy guarantees the claimed performance.

The weighting strategy assigns a *weighting factor* α_r^i to each set X_r^i where $0 \leq r \leq i \leq k-1$. If the space is discrete then the weight assigned to the point $x \in X_r^i$ on round i, is the value of the point times α_r^i times a constant normalization factor that makes the total weight be one (the definition of the weighting for non-discrete spaces is given in the statement of Theorem 2.2.1). The weighting factor is defined recursively as follows:

$$\alpha_r^{k-1} = \begin{cases} 1 & \text{if } r = \lfloor \frac{k}{2} \rfloor \\ 0 & \text{otherwise} \end{cases} ,$$

and for $0 \leq i \leq k-2$:

$$\alpha_r^i = \left(\frac{1}{2} - \gamma\right) \alpha_r^{i+1} + \left(\frac{1}{2} + \gamma\right) \alpha_{r+1}^{i+1} .$$

Recall that γ is a parameter of the majority-vote game that is fixed before the game starts.

Solving the induction, it is easy to verify that α_r^i has the following binomial distribution.

$$\alpha_r^i = \begin{cases} 0 & \text{if } r \leq i - \frac{k}{2} \\ \binom{k-i-1}{\lfloor \frac{k}{2} \rfloor - r} (\frac{1}{2} + \gamma)^{\lfloor \frac{k}{2} \rfloor - r} (\frac{1}{2} - \gamma)^{\lceil \frac{k}{2} \rceil - i - 1 + r} & \text{if } i - \frac{k}{2} < r \leq \frac{k}{2} \\ 0 & \text{if } r > \frac{k}{2} \end{cases} , \tag{2.1}$$

where we define $\binom{0}{0}$ to be 1.

The performance of our weighting strategy is given in the following theorem:

Theorem 2.2.1: *For any probability space $\langle X, \Sigma, V \rangle$ and any $\epsilon, \gamma > 0$, if the weightor plays the majority-vote game for k iterations, where k satisfies*

$$\sum_{j=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{j} (\frac{1}{2} + \gamma)^j (\frac{1}{2} - \gamma)^{k-j} \leq \epsilon , \tag{2.2}$$

and uses the following weighting in[3] *iteration* i

$$\textit{For any set } A \textit{ in the } \sigma\textit{-algebra } \Sigma \tag{2.3}$$

$$W(A) = \sum_{r=0}^{i} V(A \cap X_r^i)\alpha_r^i / Z_i \ , \tag{2.4}$$

$$\textit{where } Z_i = \sum_{r=0}^{i} V(X_r^i)\alpha_r^i \ ,$$

then the reward at the end of the game is at least $1-\epsilon$*, independent of the strategy used by the chooser.*

Before proving the theorem, we define the function β_r^i over $0 \le r \le i \le k$ which we call the "potential" of the set X_r^i. As we shall see the potential of X_r^i predicts, in some sense, the fraction of points in X_r^i that will end up in the loss set. As at the end of the game we know which points are in the loss set and which are in the reward set, it is reasonable to define the potential for $i = k$ as

$$\beta_r^k = \begin{cases} 0 & \text{if } r > \frac{k}{2} \\ 1 & \text{if } r \le \frac{k}{2} \end{cases} \ . \tag{2.5}$$

For $i < k$ we define the potential recursively:

$$\beta_r^i = \left(\frac{1}{2} - \gamma\right)\beta_r^{i+1} + \left(\frac{1}{2} + \gamma\right)\beta_{r+1}^{i+1} \ . \tag{2.6}$$

A closed form formula for β_r^i is the tail of the binomial distribution:

$$\beta_r^i = \begin{cases} 1 & \text{if } r \le i - \frac{k}{2} \\ \sum_{j=0}^{\lfloor \frac{k}{2} \rfloor - r} \binom{k-i}{j} (\frac{1}{2}+\gamma)^j (\frac{1}{2}-\gamma)^{k-i-j} & \text{if } i - \frac{k}{2} < r \le \frac{k}{2} \\ 0 & \text{if } r > \frac{k}{2} \end{cases} \ . \tag{2.7}$$

The weight factor function, α_r^i, is in some sense a discrete derivative of the potential function along the r axis:

$$\alpha_r^i = \beta_r^{i+1} - \beta_{r+1}^{i+1} \ . \tag{2.8}$$

The main property of the weighting scheme is that it guarantees that the average potential does not increase at any step. This property is proved in the following lemma.

Lemma 2.2.2: *If the weighting scheme described in Equation (2.3) is used by the weightor, then*

$$\beta_0^0 \ge \sum_{r=0}^{1} q_r^1 \beta_r^1 \ge \sum_{r=0}^{2} q_r^2 \beta_r^2 \ldots \ge \sum_{r=0}^{k} q_r^k \beta_r^k \ ,$$

for any strategy of the chooser.

[3] In the special case where X is discrete, it is sufficient to define the weight of each point. In this case we set $W(x) = \alpha_r^i V(x)$ for all $x \in X_r^i$.

Proof of Lemma 2.2.2: Recall that $q_r^i = V(X_r^i)$ and $x_r^i = V(M \cap X_r^i)$. At each iteration i the adversary chooses the variables $0 \leq x_r^i \leq 1$, and we get the following formula for the transition to the next iteration:

$$\begin{aligned} q_r^{i+1} = q_{r-1}^i x_{r-1}^i + q_r^i(1 - x_r^i) \quad &\text{for} \quad 1 \leq r \leq i \ , \\ q_0^{i+1} = q_0^i(1 - x_0^i) \quad &\text{for} \quad r = 0 \ , \\ q_{i+1}^{i+1} = q_i^i x_i^i \quad &\text{for} \quad r = i+1 \ . \end{aligned} \tag{2.9}$$

Using this we can get a formula that relates the sum $\sum_{r=0}^{i} q_r^i \beta_r^i$ for consecutive iterations.

$$\sum_{r=0}^{i+1} q_r^{i+1} \beta_r^{i+1} = q_0^i(1 - x_0^i)\beta_0^{i+1} + \sum_{r=1}^{i} [q_{r-1}^i x_{r-1}^i + q_r^i(1 - x_r^i)]\beta_r^{i+1} + q_i^i x_i^i \beta_{i+1}^{i+1} \ .$$

and rearranging the sum gives us that

$$\sum_{r=0}^{i+1} q_r^{i+1} \beta_r^{i+1} = \sum_{r=0}^{i} q_r^i [(1 - x_r^i)\beta_r^{i+1} + x_r^i \beta_{r+1}^{i+1}] = \sum_{r=0}^{i} q_r^i \beta_r^{i+1} + \sum_{r=0}^{i} q_r^i x_r^i (\beta_{r+1}^{i+1} - \beta_r^{i+1}) \ . \tag{2.10}$$

On the other hand, from the weight restriction we get:

$$\sum_{r=0}^{i} W(M_r^i) \geq \frac{1}{2} + \gamma \ ,$$

and as $M_r^i \subseteq X_r^i$ the definition of the weight function gives:

$$\frac{1}{Z_i} \sum_{r=0}^{i} V(M_r^i)\alpha_r^i = \frac{1}{Z_i} \sum_{r=0}^{i} V(M_r^i)(\beta_r^{i+1} - \beta_{r+1}^{i+1}) \geq \frac{1}{2} + \gamma \ .$$

Using the fact that $\beta_r^{i+1} > \beta_{r+1}^{i+1}$ and the definition of Z_i we get that

$$\sum_{r=0}^{i} q_r^i x_r^i (\beta_{r+1}^{i+1} - \beta_r^{i+1}) \leq (\frac{1}{2} + \gamma) \sum_{r=0}^{i} q_r^i (\beta_{r+1}^{i+1} - \beta_r^{i+1}) \ . \tag{2.11}$$

Substituting (2.11) into the RHS of (2.10) we finally get that

$$\sum_{r=0}^{i+1} q_r^{i+1} \beta_r^{i+1} \leq \sum_{r=0}^{i} q_r^i \beta_r^{i+1} + (\frac{1}{2} + \gamma) \sum_{r=0}^{i} q_r^i (\beta_{r+1}^{i+1} - \beta_r^{i+1}) =$$

$$\sum_{r=0}^{i} q_r^i ((\frac{1}{2} + \gamma)\beta_{r+1}^{i+1} + (\frac{1}{2} - \gamma)\beta_r^{i+1}) = \sum_{r=0}^{i} q_r^i \beta_r^i \ .$$

The last equality is a based on Equation (2.6). ∎

Proof of Theorem 2.2.1 From Equation (2.7) it is immediate that the LHS of Equation 2.2 is equal to β_0^0, thus, by choice of k, $\beta_0^0 \leq \epsilon$, which means that the initial expected potential is small. Combining this with the inequality from Lemma 2.2.2, that implies that the potential never increases, we get that

$$\epsilon \geq \beta_0^0 \geq \sum_{r=0}^{k} \beta_r^k q_r^k \ .$$

On the other hand, from the Equation (2.5) we have:

$$\sum_{r=0}^{k} \beta_r^k q_r^k = \sum_{r=0}^{\lfloor \frac{k}{2} \rfloor} q_r^k = V(L) \ .$$

Thus the value of the loss set L is at most ϵ. ∎

In order to see that the result given in Theorem 2.2.1 is meaningful, we give an explicit choice for k that is close to the optimal choice for small ϵ and γ.

Corollary 2.2.3: *Theorem 2.2.1 holds if the number of iterations is chosen to be* $k = \frac{1}{2}(\frac{1}{\gamma})^2 \ln \frac{1}{\epsilon}$.

Proof: The LHS of Inequality 2.2 is equal to the probability of the tail of a binomial distribution with $p = 1/2 + \gamma$ and k trials, and can be bounded by Chernoff bounds [Bollobas, 1985, page 11]. as follows.

$$P(S_{k,p} \leq \lfloor \frac{k}{2} \rfloor) \leq P(S_{k,p} \leq \frac{k}{2}) \leq \exp(kH(\frac{k/2}{k}))$$

where

$$H(\frac{1}{2}) = \frac{1}{2} \ln \frac{1/2+\gamma}{1/2} + \frac{1}{2} \ln \frac{1/2-\gamma}{1/2} = \frac{1}{2} \ln(1-(2\gamma)^2)$$

plugging this into the previous formula, and requiring the tail to be bounded by ϵ we get

$$\epsilon \geq \exp(\frac{k}{2} \ln(1-(2\gamma)^2))$$

$$\ln \epsilon \geq \frac{k}{2} \ln(1-(2\gamma)^2)$$

$$k \geq 2 \frac{\ln(\epsilon)}{\ln(1-(2\gamma)^2)}$$

using the bound $\ln(1-x) < -x$ for $0 \leq x \leq 1$ we get that for the above to hold it suffices to require the following

$$k \geq 2 \frac{\ln(1/\epsilon)}{(2\gamma)^2} = \frac{1}{2} \frac{1}{\gamma^2} \ln(\frac{1}{\epsilon})$$

∎

2.2.1 Optimality of the weighting scheme

We shall now show that, in some natural cases, the weighting strategy devised in this section is optimal. Assume the probability space $\langle X, \Sigma, V \rangle$ has the property that for any measurable set $A \in \Sigma$ there exists another set $A' \in \Sigma$ such that $V(A') = \frac{1}{2} V(A)$. Let us call such a probability space "divisible". One natural example of a divisible space is the Euclidean space $X = R^n$, where Σ is the Borel algebra over R^n and the measure V is a density measure that assigns all single points a value of zero.

In this case there is a simple strategy for the chooser such that for any strategy of the weightor the size of the reward set after k iterations will be at most

$$\sum_{j=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{j} (\frac{1}{2}+\gamma)^j (\frac{1}{2}-\gamma)^{k-j} \ . \tag{2.12}$$

Which means that the choice of k in Equation 2.2 of Theorem 2.2.1 is the smallest possible. The idea of the strategy is for the chooser to decide whether or not to mark each point in each iteration independently at random with probability $1/2+\gamma$ of choosing to mark it. This way the probability that any point is marked in more than half of the iterations is equal to the binomial tail of Equation (2.12). However, selecting each point in a continuous domain independently at random is ill defined, and a way for selecting measurable sets with similar properties is required.

In order to prove the existence of the optimal chooser strategy we need the following simple lemma, regarding divisible probability spaces

Lemma 2.2.4: *Suppose that the probability space $\langle X, \Sigma, V\rangle$ is divisible, and that W is another probability measure defined on $\langle X, \Sigma\rangle$.*

Then for any set $A \in \Sigma$ there exists a set $A' \subset A, A' \in \Sigma$ such that $V(A') = (1/2+\gamma)V(A)$ and $W(A') \geq (1/2+\gamma)W(A)$

The proof of this lemma is given in the appendix.

The strategy is defined as follows. In the ith iteration, the space X is divided into 2^{i-1} sets F_j, such that two points in X belong to the same set if they have been marked in exactly the same iterations in the past. Each of these sets is in Σ, thus, for each set F_j, there exists a corresponding set $F'_j \in \Sigma$, such that $V(F_j) = (1/2+\gamma)V(F'_j)$, and $W(F_j) \geq (1/2+\gamma)W(F'_j)$. The chooser marks the points in the set $\cup_j F'_j$. It is easy to check that this is a legitimate marking. It is also not hard to see that the value of the loss set is equal to the probability that a random coin, whose probability of "heads" is $1/2+\gamma$, will fall "tails" more than $k/2$ times in k flips. We thus get a legitimate strategy that behaves essentially like a strategy that marks each point independently at random with probability $1/2+\gamma$ and the claim easily follows.

2.2.2 The representational power of majority gates

Our analysis of the majority-vote game can be used to prove an interesting result regarding the representation of Boolean functions as a majority over other Boolean functions. This application of boosting has been discovered by Schapire [Schapire, 1992]. A slightly weaker version of this result was independently proven by Goldmann, Hastad, and Razborov [Goldmann *et al.*, 1992] using a completely different proof technique. In the following presentation we follow their notation.

Let f denote a Boolean function whose domain is $\{-1,1\}^n$ and range is $\{-1,1\}$. Let H be a set of Boolean functions defined over the same domain and range. We use $\mathcal{D}$ to denote a distribution over the domain $\{-1,1\}^n$. Let the correlation between f and H with respect to $\mathcal{D}$ be defined as

$$D_H^{\mathcal{D}}(f) \doteq \max_{h\in H} E_{\mathcal{D}}[f(x)h(x)] \ .$$

The distribution-free correlation between f and H is defined as

$$D_H(f) \doteq \min_{\mathcal{D}} D_H^{\mathcal{D}}(f) \ .$$

The majority function is defined as follows

$$MAJ(x_1,\ldots,x_k) = \text{sign}\left(\sum_{i=1}^{k} x_i\right) \ ,$$

where

$$\text{sign}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} .$$

Using our boosting algorithm we prove the following result

Theorem 2.2.5: *Let f be a Boolean function over $\{-1,1\}^n$ and H be a set of functions over the same domain. Then if $k > 2\ln(2)nD_H^{-2}(f)$, then f can be represented as*

$$f(x) = MAJ(h_1(x), \ldots, h_k(x)) ,$$

where $h_i \in H$.

Proof: Assume the majority-vote game is played over the domain $\{-1,1\}^n$ and that the value of a set is the number of points in it divided by 2^n. Assume the chooser in the majority-vote game chooses which points to mark by selecting a function $h \in H$ such that $\Pr_{\mathcal{D}}(h(x) \neq f(x)) \leq 1/2 - \gamma$ and marking all x such that $h(x) = f(x)$. By definition of $D_H(f)$, such a function exists for every distribution $\mathcal{D}$ if $\gamma = D_H(f)/2$. Theorem 2.2.1 provides us with a method for selecting the distributions $\mathcal{D}_i$, which correspond to the the weightings W_i. This selection guarantees that the majority over the corresponding hypotheses will be very close to f. More specifically, it guarantees that if $k = 1/2\gamma^{-2}\ln 1/\epsilon$, the number of points $x \in \{-1,1\}^n$ such that $MAJ(h_1(x), \ldots, h_k(x)) \neq f(x)$ is smaller than $\epsilon 2^n$, by setting $\epsilon < 2^{-n}$ we guarantee that $MAJ(h_1(x), \ldots, h_k(x)) = f(x)$ for all $x \in \{-1,1\}^n$. Plugging our selection for γ and ϵ into $k = 1/2\gamma^{-2}\ln\epsilon$ we finish the proof. ∎

Goldmann, Hastad and Razborov ([Goldmann *et al.*, 1992]), prove Theorem 2.2.5 using a elegant application of von Neumann's Min-Max Theorem. They consider the selection of the $h \in H$ that is correlated to f, as the following game. One side in the game, which we call the weightor, defines the probability $\mathcal{D}$ over X, and the other side,the chooser, chooses $h \in H$. Unlike our majority-vote game, this game consists of a single trial of this kind, and the players are given the knowledge of each others decision only *after* each of them makes its own decision. The chooser's gain (and the weightor loss) is the correlation of f and h according to $\mathcal{D}$, $E_{\mathcal{D}}[f(x)h(x)] \geq 1/2 + \gamma$.

The so-called "simple" strategy for the chooser is to select a single function $h \in H$ all of the time, and the simple strategy for the weightor is to always select a single distribution. The assumption of the theorem implies that for any simple strategy of the weightor there exists a simple strategy for the chooser that guarantees it a gain of at least $1/2 + \gamma$. The Min-Max theorem implies that because of this, there is a *mixed* strategy for the chooser that would have an expected gain of at least $1/2 + \gamma$ against *any* strategy of the weightor. A mixed strategy is a distribution over the simple strategies. In other words, there exists a distribution over H such that if h is chosen according to this distribution, then the expected correlation between h and f with respect to *any* distribution over X is larger than $1/2 + \gamma$. In particular, if the distribution $\mathcal{D}$ is concentrated on any particular element $x_0 \in X$, then the probability that $h(x_0) = f(x_0)$ is larger than $1/2 + \gamma$. If we choose k functions independently at random from H according to this distribution, we can make the probability that $MAJ(h_1(x_0), \ldots, h_k(x_0)) = f(x_0)$ arbitrarily high. As X is a finite space, a sufficiently large value of k guarantees that, with high probability, the majority vote over k random functions is correct on all of X. Using the fact that $|X| = 2^n$ and Chernoff bounds, they get the statement of the theorem.

This proof is very short and elegant. However, it is not a constructive proof. On the other hand, our proof is constructive in that it shows how to generate the distributions that correspond to the desired functions in H.

For completeness we give a simple lemma (Lemma 3.2 in [Goldmann *et al.*, 1992]) that gives an approximate converse to Theorem 2.2.5.

Lemma 2.2.6: *Let f and H be as in Theorem 2.2.5. Then if f can be represented as*

$$f(x) = MAJ(h_1(x), \ldots, h_k(x)) \ ,$$

where $h_i \in H$, then $D_H(f) \geq 1/k$.

Proof: From the definition of the majority function and the requirement that the argument of the sign function is never zero, we get that for every $x \in \{-1,1\}^n$, there are at least $(k+1)/2$ indices i such that $h_i(x) = f(x)$. Fixing any distribution $\mathcal{D}$ over $\{-1,1\}^n$, we get that

$$\sum_{i=1}^{k} \Pr_{x \in \mathcal{D}} \left(h_i(x) = f(x)\right) = \sum_{x \in \{-1,1\}^n} \Pr_{\mathcal{D}}(x) \left|\{1 \leq i \leq k \mid h_i(x) = f(x)\}\right| \ \geq \ (k+1)/2 \ .$$

The pigeon-hole principle guarantees that there exists at least one index $1 \leq i \leq k$ such that $\Pr_{x \in \mathcal{D}}(h_i(x) = f(x)) \geq (k+1)/2$. This implies that $D_H^{\mathcal{D}}(f) \geq 1/k$. As this holds for all $\mathcal{D}$, we get the statement of the lemma. ∎

2.3 Boosting a weak learner using a majority vote

In this section we shall describe the connection between the majority-vote game and the problem of boosting a weak learning algorithm.

We start by presenting the minimal notation that is needed for analyzing our boosting algorithms. We then present our algorithms and their analysis. Later, in Section 2.3.5, we give a more complete notational framework, and use this framework to relate our results to other results in PAC learning theory.

2.3.1 Preliminaries

We start by giving the definitions of a minimal framework of distribution-free concept learning that is needed for presenting our main results. A *concept* is a binary-valued mapping over some domain X. We denote use the letter c to denote a concept and $c(x)$ to denote the label of the instance x according to the concept c. A *concept class* $\mathbf{C}$ is a collection of concepts.[4]

The learners task is to learn an approximation to a concept c. The learner knows *a-priori* that the concept is in some known class $\mathbf{C}$, but has no prior knowledge of the specific choice of $c \in \mathbf{C}$. The learner is assumed to have access to a source **EX** of examples. Each time **EX** is called, one instance is randomly

[4] In order to define *polynomial* PAC learnability, the complexity of the sample space and of the concept class need to be parameterized. In our initial basic setting we suppress this parameterization and the issue of polynomial versus non polynomial learning, we return to fully discuss this issue in Section 2.3.5.

and independently chosen from X according to some fixed but unknown and arbitrary distribution $\mathcal{D}$.[5] The oracle returns the chosen instance $x \in X$, along with its label according to the concept c, which is denoted $c(x)$. Such a labeled instance is called an *example*. We assume **EX** runs in unit time.

Given access to **EX** the learning algorithm runs for some time and finally outputs an *hypothesis* h. The hypothesis is a description of an algorithm (possibly probabilistic) that receives as input an instance $x \in X$ and generates a binary output. This output is called the "prediction" of the hypothesis for the label $c(x)$. We write $P(h(x) = c(x))$ to indicate the probability, over the distribution $\mathcal{D}$ on X and random coin flips of the hypothesis, that the hypothesis correctly predicts the labels of the concept c. This probability is called the *accuracy* of the hypothesis h. The probability $P(h(x) \neq c(x))$ is called the *error* of h with respect to c under $\mathcal{D}$; if the error is no more than ϵ, then we say h is ϵ-good with respect to the target concept c and the distribution $\mathcal{D}$.

We say that a learning algorithm A has a *uniform* sample complexity $m(\epsilon, \delta)$ if it achieves the following performance. For all $0 < \epsilon, \delta < 1$, all $\mathcal{D}$, and all $c \in \mathbf{C}$, when given parameters ϵ and δ, algorithm A makes at most m calls to **EX** and outputs a hypothesis h that with probability at least $1 - \delta$ is an ϵ-good approximation of c under $\mathcal{D}$. Similarly we define the time and space complexity of A to be functions that bound the time and space required by A and denote them by $t(\epsilon, \delta)$ and $s(\epsilon, \delta)$ respectively. If a learning algorithm cannot achieve some values of ϵ and δ, or if the resources required for achieving these values are not uniformly bounded for all distributions and concepts, we define $m(\epsilon, \delta), t(\epsilon, \delta)$ and $s(\epsilon, \delta)$ to be infinite for these values.

The concept of a *boosting* algorithm was first presented by Schapire in [Schapire, 1990]. A boosting algorithm is a learning algorithm that uses as a subroutine a different learning algorithm. The goal of the boosting algorithm is to efficiently generate high-accuracy hypotheses using a learning algorithm that can efficiently generate only low-accuracy hypotheses. The boosting algorithm invented by Schapire [Schapire, 1990], was a breakthrough in that it showed that any polynomial time learning algorithm that generates hypotheses whose error is just slightly smaller than 1/2 can be transformed into a polynomial time learning algorithm that generates hypotheses whose error is arbitrarily small. The boosting algorithms presented in this paper achieve better performance than those presented by Schapire and the resulting hypotheses are simpler. A comparison of the performance of the algorithms is given in Section 2.3.5.

We use the generic name **WeakLearn** to refer to the learning algorithm whose performance we wish to boost, and we refer to those hypotheses generated by **WeakLearn** that have the guaranteed accuracy as *weak* hypotheses. We assume that there exist some real values $0 \leq \epsilon_0 < 1/2$ and $0 \leq \delta_0 < 1$ such that **WeakLearn**, given m_0 examples labeled according to some concept $c \in \mathbf{C}$, generates a hypothesis whose error is smaller than ϵ_0 (i.e. a weak hypothesis) with probability larger than $1 - \delta_0$ over the distribution of the training examples. We denote by m_0, t_0, and s_0 uniform upper bounds on the sample size, time, and space required by **WeakLearn** to achieve this accuracy. The boosting algorithms that we shall describe are able to generate hypotheses of arbitrary accuracy ϵ with arbitrarily high reliability $1 - \delta$.

[5] More formally, we assume that $\langle X, \Sigma, \mathcal{D} \rangle$ is a probability space, and that $\mathbf{C}$ is a set of functions that are measurable with respect to Σ. Moreover, we assume that all subsets of X that are considered in this paper are measurable with respect to Σ.

The parameters ϵ_0 and δ_0 measure the discrepancy between the performance of **WeakLearn** and the performance of an "ideal" learning algorithm that always generates a hypothesis that has no error with respect to the target concept. The performance of the weak learning algorithms that we discuss is extremely poor. They are almost completely unreliable, and even when they succeed, they output a hypothesis whose error is close to that of a random guess. We thus find it useful to define two new quantities $\gamma = 1/2 - \epsilon_0$ and $\lambda = 1 - \delta_0$. These parameters measure how far the learning algorithm is from a completely useless algorithm and arise naturally in the design and analysis of our boosting algorithms. We shall show that the resources required by our algorithms are uniformly bounded by functions whose dependence on $1/\gamma, 1/\lambda, 1/\epsilon$, and $1/\delta$ is either logarithmic or low-order polynomial.

For the main part of our analysis, in Sections 2.3.2 and 2.3.3, we restrict ourselves to boosting deterministic learning algorithms that generate deterministic hypotheses. Later, in Section 2.3.4, we show that all of our algorithms and their analysis hold, with very little change, for the case that the learning algorithm and the resulting hypotheses are randomized.

2.3.2 Boosting using sub-sampling

One simple way of applying the results of the majority-vote game to boost the performance of **WeakLearn** is by using it to find a small hypothesis that is consistent with a large set of training examples. The algorithm $\mathbf{B}_{\text{Samp}}$, which is summarized in Figure 2.3, is based on this principle.

The first step of $\mathbf{B}_{\text{Samp}}$ is to collect a training set. Formally, this means making m calls to **EX**, generating the set $S = \{(x_1, l_1), \ldots, (x_m, l_m)\}$.[6] The goal of boosting is to generate a hypothesis that is correct on all examples in S.

As the sample is a finite set of size m, the requirement that a hypothesis is correct on all points in the sample is equivalent to the requirement that the hypothesis has error smaller than $1/m$ with respect to the uniform distribution on the sample. In order to do that, $\mathbf{B}_{\text{Samp}}$ generates different distributions on the training sample, and each time calls **WeakLearn** to generate a weak hypothesis, that is, a hypothesis that has error smaller than $1/2 - \gamma$ with respect to the given distribution. Each different distribution forces **WeakLearn** to generate a weak hypothesis whose errors are on different sample points.[7] The goal of the boosting algorithm is to control the location of these errors in such a way that after a small number of weak hypotheses have been generated, the majority vote over all weak hypotheses will give the correct label on each point. In other words, for each point in S, the fraction of the weak hypotheses that assign the point with the correct label is larger than half.

The problem of generating these distributions is equivalent to the problem of the booster in the majority-vote game described in the previous section, under the following correspondence of terms. The *value* of a point corresponds to the probability assigned to the point by the target distribution (the uniform distribution in our case). The *weight* of a point corresponds to the probability assigned to it by the boosting algorithm. The decision of the adversary to *mark* a point corresponds to the decision by **WeakLearn** to

[6] In many actual machine learning scenarios, the training set S is the basic input to the learning algorithm, and thus this step is only formal.

[7] Ignoring, for a moment, the fact that **WeakLearn** has probability δ_0 of failing to generate a weak hypothesis.

Algorithm $\mathbf{B}_{\text{Samp}}$
Input: **EX**,**WeakLearn**, γ, m
Output: A hypothesis that is consistent on a random sample of size m.

1. Call **EX** m times to generate a sample $S = \{(x_1, l_1), \ldots, (x_m, l_m)\}$. To each example (x_j, l_j) in S there is a corresponding *weight* w_j and *count* r_j. Initially, all weights are $1/m$ and all counts are zero.
2. Find a (small) k that satisfies

$$\sum_{i=\lceil k/2 \rceil}^{k} \binom{k}{i} (1/2 - \gamma)^i (1/2 + \gamma)^{k-i} < \frac{1}{m}$$

   (For example, any $k > 1/(2\gamma^2) \ln m$ is sufficient.)
3. Repeat the following steps for $i = 1 \ldots k$.
   (a) repeat the following steps for $l = 1 \ldots (1/\lambda) \ln(2k/\delta)$ or until a good hypothesis is found.
      i. Call **WeakLearn**, referring it to **FiltEX** as its source of examples, and save the returned hypothesis as h_i.
      ii. Sum the weights of the examples on which $h_i(x_j) \neq l_j$. If the sum is smaller than $1/2 - \gamma$ then declare h_i a weak hypothesis and exit the loop.
   (b) Increment r_j by one for each example on which $h_i(x_j) = l_j$.
   (c) Update the weights of the examples according to $w_j = \alpha_{r_j}^i$, where α_r^i is defined in Formula (2.1).
   (d) Normalize the weights by dividing each weight by $\sum_{j=1}^{m} w_j$.
4. Return as the final hypothesis, h_M, the majority vote over $h_1, \ldots, h_k$.

Subroutine FiltEX

1. choose a real number x uniformly at random in the range $0 \leq x < 1$.
2. Perform a binary search for the index j for which

$$\sum_{i=1}^{j-1} w_i \leq x < \sum_{i=1}^{j} w_i$$

   ($\sum_{i=1}^{0} w_i$ is defined to be zero.)
3. Return the example (x_j, l_j)

Figure 2.3: A description of the algorithm for boosting by sub-sampling

generate a weak hypothesis that is correct on the point. The *reward set* corresponds to the set on which the majority vote over the weak hypotheses is correct and the *loss* is the probability that the majority makes a mistake, measured with respect to the target distribution. This correspondence lies in the center of the analysis of algorithm $\mathbf{B}_{\text{Samp}}$.

Before we give the first theorem regarding the performance of $\mathbf{B}_{\text{Samp}}$ we must address the fact that **WeakLearn** is not guaranteed to always generate a weak hypothesis. This event is only guaranteed to happen with probability λ. However, it is easy to check the hypothesis returned by **WeakLearn** and calculate its error on the sample. If this error is larger than $\epsilon_0 = 1/2 - \gamma$, **WeakLearn** is called again, using a different subset of the examples in S.[8] This is the role of statement 3.a.ii of $\mathbf{B}_{\text{Samp}}$. However, this test has non-zero probability of failing any arbitrary number of times. In order to guarantee that the boosting algorithm has uniform finite running time, $\mathbf{B}_{\text{Samp}}$ tests only a pre-specified number of hypotheses. As we shall show in the second part of the proof of Theorem 2.3.3, the probability that all these hypotheses will have error larger than ϵ_0 is smaller than $\delta/2$. The following theorem shows that if all k iterations manage to find a weak hypothesis, then the final hypothesis generated by $\mathbf{B}_{\text{Samp}}$ is consistent with all the labels in the sample.

Theorem 2.3.1: *If all the hypotheses that are used by algorithm* $\mathbf{B}_{\text{Samp}}$ *are* ϵ_0 *accurate, then the hypothesis* h_M*, output by* $\mathbf{B}_{\text{Samp}}$*, is consistent on the sample* S*.*

Proof: From the correspondence with the majority-vote game defined above, and from Theorem 2.2.1, we get that the error of the hypothesis output by $\mathbf{B}_{\text{Samp}}$ is smaller than $1/m$, As the target distribution is uniform it assigns each point in S with probability $1/m$. Thus the output hypothesis must be correct on all points in S. ∎

Two issues remain in order to show that $\mathbf{B}_{\text{Samp}}$is an effective learning algorithm. First, we need to show that there is a way for selecting m, the size of the sample S, so that the hypotheses generated by $\mathbf{B}_{\text{Samp}}$, that is guaranteed to be consistent on S, will also be have a small probability of error on a random example outside of S. Second, we need to show that the algorithm uses uniformly bounded resources.

The fact that using a large enough sample guarantees that a consistent hypothesis will have small error on the whole domain stems from the fact that k, the number of hypotheses that are combined by the majority rule, increases like $O(\log|S|)$, as was proven in Corollary 2.2.3. Before getting into a detailed proof, let us give a rough sketch of a proof for a simple special case. Assume that the hypotheses generated by **WeakLearn** are chosen from a *finite* set of hypotheses H. Denote the set of hypotheses generated by $\mathbf{B}_{\text{Samp}}$ by H_M. The size of H_M is $|H|^{c\log m}$, where $c = 1/(2\gamma^2)$. Following the well-known analysis of the Occam's razor principle [Blumer *et al.*, 1987] we get that the probability that the final hypothesis is consistent with a random sample of size m but has error larger than ϵ is smaller than $|H_M|(1-\epsilon)^m = |H|^{c\log m}(1-\epsilon)^m$. This quantity decreases rapidly with m. In particular, selecting m large enough that $m \geq (1/\epsilon)(\log(1/\delta) + (1/2\gamma^2)\log m \log|H|)$, guarantees that the hypothesis will have error smaller than ϵ with probability larger than $1-\delta$.

[8] Note that as **WeakLearn** is guaranteed to succeed with probability at least λ on *any* distribution over the sample space, it is guaranteed to succeed on the uniform distribution over S.

Although this simple analysis gives the correct orders of magnitude, it is incomplete in that it depends on the size of H. In many cases this size is very large, moreover, often H is infinite or even uncountable. These cases can be analyzed using the notion of VC-dimension. However, Schapire [Schapire, 1990], suggested the following elegant proof that is based only on the assumption that the size of the sample used by **WeakLearn** is uniformly bounded. Although the final hypothesis is guaranteed to be consistent with the whole sample, which is of size m, the number of examples from the sample that are ever used by **WeakLearn** is[9] $O(\log m)$. In other words, for large m only a small fraction of the training examples are ever used by **WeakLearn**!

This small subset of S can be seen as a *representation* of the final hypothesis, h_M. Instead of saving the hypotheses generated by **WeakLearn**, the boosting algorithm can save the set of examples that were returned from **FiltEX** during the run of each algorithm. Later, when the value of $h_M(x)$ has to be calculated on some new example x, the weak learning algorithms can be run again, using the saved sets of examples, to regenerate the weak hypotheses, and using these weak hypotheses $h_M(x)$ can be reconstructed.

Littlestone, Warmuth and Floyd [Littlestone and Warmuth, 1986, Floyd and Warmuth, 1993] have analyzed algorithms that represent their hypotheses as sets of examples. As the above observation is of independent interest in the context of their work, we state it as a theorem.

Theorem 2.3.2: *Let* $\mathbf{C}$ *be a concept class that is PAC learnable, then there exists a pair of polynomial-time algorithms* P *and* R *,that stand for "compress" and "reconstruct", that compute the following mappings. Let* $S = \{(x_1, c(x_1)), \ldots, (x_m, c(x_m))\}$ *be a sample labeled according to some concept* $c \in \mathbf{C}$*. Then, for all such* S*,*

- $P(S) = \{(x_{i_1}, c(x_{i_1})), \ldots, (x_{i_r}, c(x_{i_r}))\}$*, such that* $1 \leq i_j \leq m$ *for all* $1 \leq j \leq r$*, i.e. the compression algorithm selects an ordered sequence of of length* r *from the examples in the sample* S*.*
- $R(P(S)) = h$ *is a hypothesis such that for every sample point* x_i *in* S*,* $h(x_i) = l_i$*. In other words, the algorithm* R *can reconstruct the labels of all the examples in* S *when given the labeled examples selected by* P*.*
- $r = O(\log m)$*.*

We now move on to prove a bound on the size of the sample that $\mathbf{B}_{\text{Samp}}$ has to use in order to guarantee that the final hypothesis has error smaller than ϵ. In the proof of this theorem we use a technique invented by Littlestone and Warmuth [Littlestone and Warmuth, 1986] that appears as Appendix A in [Floyd and Warmuth, 1993].

Theorem 2.3.3: *Let* **WeakLearn** *be a deterministic learning algorithm that generates, with probability* $\lambda > 0$ *over the random training examples with which it is trained, a deterministic hypothesis whose error is smaller than* $1/2 - \gamma$*, for some* $\gamma > 0$*. Assume the number of training examples required to achieve this is uniformly bounded by* m_0*. Then the hypothesis* h_M *generated by* $\mathbf{B}_{\text{Samp}}$ *has the following property.*

For any $\epsilon, \delta > 0$*, if* $\mathbf{B}_{\text{Samp}}$ *uses a sample of size at least* m*, where*

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{2}{\delta} + \frac{m_0}{2}\left(\frac{\ln m + 1}{\gamma}\right)^2\right) \ ,$$

[9] Here we fix the concept class and its VC dimension d. If the VC dimension is not fixed, the $O(d \log m)$ examples are required.

then the probability that h_M has error larger than ϵ is smaller than δ. Where the probability is defined over the random choice of the sample S and over the internal random coin flips in $\mathbf{B}_{\text{Samp}}$.

Proof: We are interested in bounding the probability of the set of samples and internal coin flips of $\mathbf{B}_{\text{Samp}}$ that generate a hypothesis that has error larger than ϵ. We do that by covering this set by two disjoint sets. The first set is the set of samples and coin flips that cause $\mathbf{B}_{\text{Samp}}$ to generate a hypothesis that is consistent with the sample and yet has error larger than ϵ. The second is the set of samples and coin flips that causes $\mathbf{B}_{\text{Samp}}$ to generate a hypothesis that is inconsistent with the sample. The first and second parts of the proof bound the probabilities of these two sets respectively.

Part 1: We want to show that there is only a small probability that a random sequence of training examples $S = \langle (x_1, l_1), \ldots, (x_m, l_m) \rangle$ labeled according to $c \in \mathbf{C}$, can cause $\mathbf{B}_{\text{Samp}}$ to generate a hypothesis that is consistent with S but has error larger than ϵ.

We first sketch the argument. We consider the following mapping of arbitrary sequences of km_0 labeled examples into hypotheses. The sequence is partitioned into k blocks of length m_0, each block is fed into **WeakLearn**. Using this block **WeakLearn** generates a hypothesis.[10] Finally, these k hypotheses are combined by a majority vote to generate a single hypothesis. We define two properties on sequences chosen out of S that are based on the hypothesis to which these sequences are mapped. The first property is that the hypothesis is consistent with all the examples in S, the second property is that the hypothesis has error larger than ϵ w.r.t. the distribution $\mathcal{D}$ and the underlying concept. We call sequences that have both properties "bad" sequences. We show that the probability of a sample S from which a bad sequence can be chosen is very small. However, if by using some sequence of coin flips, $\mathbf{B}_{\text{Samp}}$ can generate a consistent hypothesis that has a large error, then there *exists* a way of choosing a bad sequence out of S, which means that the probability of $\mathbf{B}_{\text{Samp}}$ generating such a hypothesis is small.

To bound the probability of samples S from which a bad sequence can be chosen, one can view the elements of S that are not in the sequence as random test points on which the hypothesis is tested. As most of the points in S are not in the sequence, it is very unlikely that the hypothesis is consistent with all these examples and yet has a large probability of making an error. This observation, together with the fact that the total number of sequences of km_0 elements from S is not too large, gives us the proof of this part of the theorem.

We now give the formal proof. Which is an adaptation of a technique used by Warmuth and Littlestone in [Littlestone and Warmuth, 1986]. Fix any concept $c \in \mathbf{C}$. Let $S = \langle (x_1, l_1), \ldots, (x_m, l_m) \rangle$ be the sequence of randomly drawn training examples returned by **EX** in step 1 of a specific run of $\mathbf{B}_{\text{Samp}}$ such that for all i, $l_i = c(x_i)$. Let $S' = \langle (x_{t_1}, l_{t_1}), \ldots, (x_{t_d}, l_{t_d}) \rangle$ denote a sequence of examples chosen out of S.

Let $\mathcal{T}$ be the collection of all m^d sequences of length $d = km_0$ of integers in $\{1 \ldots m\}$. For any sequence of examples $S = \langle (x_1, l_1), \ldots, (x_m, l_m) \rangle$ and for any $T \in \mathcal{T}$ we denote $\langle (x_{t_1}, l_{t_1}), \ldots, (x_{t_d}, l_{t_d}) \rangle$ by S'_T. We denote the hypothesis to which this sequence is mapped by the mapping defined above by $h_M(S'_T)$.

Fixing T, let U_T be the set of all sequences of examples S such that the hypothesis $h_M(S'_T)$ has error larger than ϵ. Recall that the error of h_M is the probability, with respect to the distribution $\mathcal{D}$, of the symmetric difference between h_M and c. Let C_T be the set of all sequences S such that $h_M(S'_T)$ is consistent

[10] we assume that **WeakLearn** is deterministic and returns a hypothesis for any sequence of m_0 examples.

with all the examples in S. Observe that each run of $\mathbf{B}_{\mathrm{Samp}}$ in which it generates a consistent hypothesis corresponds to a sequence of indices T such that C_T contains the training set S that was used by the algorithm. If $\mathbf{B}_{\mathrm{Samp}}$ has non-zero probability of generating a consistent hypothesis that has a large error when using the sample S, then there must exist some $T \in \mathcal{T}$ such that $S \in C_T \cap U_T$. We can thus upper bound the probability of failure over the random choice of S, by requiring that

$$\sum_{T \in \mathcal{T}} P^m(C_T \cap U_T) \leq \delta/2 \ .$$

For any particular $T \in \mathcal{T}$, there exists $T' \in \mathcal{T}$ where all the elements of T' are in the range $1 \ldots d$ such that $P^m(C_T \cap U_T) = P^m(C_{T'} \cap U_{T'})$. That is because the elements of S are drawn independently at random, so that any permutation of the elements in S has the same probability, and there is always a permutation of the elements of S that transforms T to T' of the desired type. It thus suffices to bound $P^m(C_T \cap U_T)$ for T of the restricted type. In this case, the choice of the hypothesis $h_M(\langle (x_{t_1}, l_{t_1}), \ldots, (x_{t_d}, l_{t_d}) \rangle)$ are only a function of the first d elements of S. If $S \in U_T$, the hypothesis has probability at least $1 - \epsilon$ of making a mistake on any of the remaining $m - d$ elements of S, thus the probability that S is in C_T, given that it is in U_T, is at most $(1 - \epsilon)^{m-d}$. Multiplying this probability by the size of $\mathcal{T}$ we get

$$m^d (1 - \epsilon)^{m-d} \leq \delta/2 \ . \tag{2.13}$$

By plugging this into $d = km_0$ we get that it is sufficient to require that

$$m^{km_0} (1 - \epsilon)^{m - km_0} \leq \frac{\delta}{2} \ ,$$

which can be translated to the following stronger requirement on m:

$$m \geq \frac{1}{\epsilon} \left(\ln \frac{2}{\delta} - km_0 (\ln m + \epsilon) \right) \ .$$

We now use $1/(2\gamma^2) \ln m$ as a choice for k, the number of weak hypotheses that are combined by **WeakLearn**. Corollary 2.2.3 shows that this choice obeys the inequality of line 2 in $\mathbf{B}_{\mathrm{Samp}}$. We thus get that it is sufficient to require that

$$m \geq \frac{1}{\epsilon} \left(\ln(2/\delta) + m_0 \frac{\ln m}{2\gamma^2} (\ln m + \epsilon) \right) \ .$$

As the statement of the theorem places a slightly stronger requirement on the minimal value of m, we get that if $\mathbf{B}_{\mathrm{Samp}}$ generates a consistent hypothesis than this hypothesis has error smaller than ϵ with probability at least $1 - \delta/2$.

Part 2: We now bound the probability that $\mathbf{B}_{\mathrm{Samp}}$ generates a hypothesis that is not consistent with the sample. From Theorem 2.3.1 we know that if all of the k hypotheses generated by **WeakLearn** have error smaller than ϵ_0 with respect to the corresponding weightings of the the sample, then the final hypothesis is consistent with the whole sample. It thus remains to be shown that for any sample S, the probability, over the random choice made in $\mathbf{B}_{\mathrm{Samp}}$ that any of the k hypotheses used by $\mathbf{B}_{\mathrm{Filt}}$ has error larger than ϵ_0 is smaller than $\delta/2k$.

Note that each time a hypothesis is returned from **WeakLearn** its error on the weighted sample is checked, and it is rejected if the error is too large. Thus the only case in which a hypothesis used by $\mathbf{B}_{\mathrm{Samp}}$ has an error larger than ϵ_0 is when all of the iterations of statement 3.a fail to generate a hypothesis with small error. As the probability that any single call to **WeakLearn** generate a good hypothesis is at least λ, the probability that all of the $(1/\lambda)\ln(2k/\delta)$ runs of **WeakLearn** performed in statement 3.a fail to generate a good hypothesis is at most

$$(1-\lambda)^{(1/\lambda)\ln(2k/\delta)} \leq \frac{\delta}{2k} \ .$$

Thus the probability that any of the k hypothesis used is not good is at most $\delta/2$. ∎

Theorem 2.3.3 gives a uniform upper bound on the sample complexity of $\mathbf{B}_{\mathrm{Samp}}$. The bound is given in terms of an implicit inequality on m, which cannot be written as an exact explicit bound. The following corollary gives an explicit upper bound on the sample complexity needed for boosting using $\mathbf{B}_{\mathrm{Samp}}$.

Corollary 2.3.4: *Let* **WeakLearn** *be a deterministic learning algorithm that generates, with probability $\lambda > 0$ over the random training examples with which it is trained, a deterministic hypothesis whose error is smaller than $1/2 - \gamma$, for some $\gamma > 0$. Assume the number of training examples required to achieve this is uniformly bounded by m_0. Then, given any $\epsilon, \delta > 0$, if $\mathbf{B}_{\mathrm{Samp}}$ is required to generate a hypothesis that is consistent with a sample of size*

$$m \geq \max\left\{208, \frac{2}{\epsilon}\ln\frac{2}{\delta}, 16\frac{m_0}{\epsilon\gamma^2}(\ln\frac{m_0}{\epsilon\gamma^2})^2\right\} \ ,$$

then with probability larger than $1-\delta$, the hypothesis output by $\mathbf{B}_{\mathrm{Samp}}$ has error smaller than ϵ.

Proof: We want to find m such that will satisfy:

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{2}{\delta} + \frac{m_0}{2}\left(\frac{\ln m + 1}{\gamma}\right)^2\right) \ ,$$

It suffices if m is larger than the maximum of twice each of the two terms in the RHS. From the first term we get $m > \frac{2}{\epsilon}\ln\frac{2}{\delta}$. To bound m w.r.t. the second term, we observe that, in general, in order to satisfy $m > a(\ln m + 1)^2$ it suffices to choose $m = 16a(\ln a)^2$, if $a \geq 5$. It thus suffices if $m > 16a(\ln a)^2 = 16 * 5 * (\ln 5)^2$, or if $m > 208$. ∎

The space requirements of $\mathbf{B}_{\mathrm{Samp}}$ are dominated by the storage of the sample. The sample size is, ignoring log factors, $\tilde{O}(1/\epsilon)$ (Corollary 2.3.4), while the storage of the hypotheses generated by **WeakLearn** is $O(k) = O(1/\gamma^2 \ \log 1/\epsilon)$.

We now discuss the time and space complexity of $\mathbf{B}_{\mathrm{Samp}}$. One easily observes that the total number of times that **WeakLearn** is called is

$$O(k\ln k) = O\left(\frac{1}{\gamma^2\lambda}\ \ln\frac{1}{\epsilon}\ \left(\ln\frac{1}{\gamma^2\delta} + \ln\ln\frac{1}{\epsilon}\right)\right) \ .$$

It is thus clear that for small values of ϵ and δ, the time complexity of $\mathbf{B}_{\mathrm{Samp}}$ is dominated by the execution of statements 3.a.ii, 3.b, 3.c and 3.d, that test and update the weights associated with the sample, and not by the running time of the learning algorithm. This time complexity is $O(m)$ and the dependence of m on ϵ, δ, and λ is given in corollary 2.3.4. The Space complexity of the algorithm is similarly dominated by the storage of the sample and its associated counters and weights in memory. The next section presents a different boosting algorithm whose space complexity is $O(\log m)$.

2.3.3 Boosting Using filtering

In the previous section we have developed one way of applying the optimal weightor strategy for the majority-vote game to the problem of boosting a weak learner. While the complexity bounds for this method are reasonably good, considerable improvement is possible in the space complexity. The space complexity of $\mathbf{B}_{\text{Samp}}$ is dominated by the storage of the training examples. In some applications the training set is in the memory anyway and this cost is taken for granted. However, in other cases (such as on-line learning), storing all the training examples in memory might be very expensive. Recall that in order to find a hypothesis with error smaller than ϵ, only $O(\log(1/\epsilon))$ out of the $O(1/\epsilon(\log(1/\epsilon))^2)$ training examples in the sample are ever used by the weak learning algorithm. In this section we present algorithms that select the examples used by **WeakLearn** in an on-line fashion from the sequence of examples supplied by **EX**. This avoids storing many examples in memory and decreases the space complexity to $O(\log(1/\epsilon))$. Selecting examples directly out of the input stream is the basis of Schapire's boosting algorithm [Schapire, 1990]. Schapire coined the term "filtering" to describe this process. The selection is viewed as a "filter" that lies between the source of examples, **EX**, and the weak learning algorithm. This filter observes each example generated by **EX** and either *rejects* it and throws it away, or *accepts* it and passes it on to **WeakLearn**.

The description of the algorithm is given in Figure 2.4. The overall structure of the algorithm is very similar to that of $\mathbf{B}_{\text{Samp}}$. The boosting algorithm generates k weak hypotheses by calling **WeakLearn** k times, each time presenting it with a different distribution over the training examples. However, while in $\mathbf{B}_{\text{Samp}}$ the examples are drawn from a set of examples that is fixed, once and for all, at the beginning of the process, in $\mathbf{B}_{\text{Filt}}$ new examples are continually drawn from the sample space by calling **EX**. Each time a new example is drawn, its weight is calculated, and a stochastic decision is made whether to accept or reject the example, such that the probability of acceptance is proportional to the weight. The proportionality constant, $\alpha^i_{\max}$, is chosen in a way that the examples with the largest weights are always accepted. Clearly, one could use any smaller proportionality factor, such as 1, without changing the distribution that is observed by **WeakLearn**. Choosing the largest possible proportionality factor maximizes the probability of accepting a random example and reduces the number of training examples required.

The analysis of $\mathbf{B}_{\text{Filt}}$ corresponds to playing the majority-vote game directly on the sample space, X, and the input distribution $\mathcal{D}$, and not on the uniform distribution over a sample, as is the case with $\mathbf{B}_{\text{Samp}}$. This simplifies the analysis with respect to the analysis of $\mathbf{B}_{\text{Samp}}$ in that there is no gap between the expected error on the training set and the expected error on a random example. On the other hand, the analysis becomes more involved as a result of the following potential problem. It might happen that during some iterations of statement (2) a large fraction of the examples generated by **EX** are rejected. As a result, the number of examples that have to be filtered in order to generate the training examples required by **WeakLearn** becomes prohibitively large. Luckily, as we shall show, the accuracy of the hypotheses that are generated by **WeakLearn** in such iterations has very little influence on the the accuracy of the final hypothesis, h_M, that is the final result of $\mathbf{B}_{\text{Filt}}$.

We use this property by defining an "abort" condition. This condition, defined at the bottom of Figure 2.4, detects iterations in which the fraction of accepted examples is small. We refer to such an event

Algorithm $\mathbf{B}_{\text{Filt}}$

Input: **EX**,**WeakLearn**, $\gamma, \lambda, \epsilon, \delta$

Output: A hypothesis h_M, that has error smaller than ϵ with probability at least $1-\delta$.

1. Find a (small) k that satisfies

$$\sum_{i=\lceil k/2 \rceil}^{k} \binom{k}{i} (1/2-\gamma/2)^i (1/2+\gamma/2)^{k-i} < \epsilon^2$$

   (For example, $k = 4/\gamma^2 \ln(1/\epsilon)$)

2. Repeat the following steps for $i = 0 \ldots k-1$, reinitializing #accept and #reject to zero each time.
   (a) Call $\mathbf{B}_{\text{Rel}}$, referring it to **FiltEX** as its source of examples, and requiring it to use **WeakLearn** to generate a hypothesis with whose error is smaller than $1/2-\gamma/2$ with probability at least $1-\delta/2k$. Save the resulting weak hypothesis as h_{i+1}.
   (b) If the abort condition happened, then define h_{i+1} to be a hypothesis that always makes a random prediction using a fair coin.
3. Return as the final hypothesis, h_M, the majority vote over $h_1, \ldots, h_k$.

Subroutine FiltEX

Repeat the following command until an example is *accepted* or until the abort condition is satisfied.

1. Call **EX**, and receive a labeled example (x, l).
2. If $i = 0$ then accept the example and return, else continue to 3.
3. Set r to be the number of indices $1 \le j \le i$ such that $h_j(x) = l$, and calculate

$$\alpha_r^i = \begin{cases} \binom{k-i-1}{\lfloor \frac{k}{2} \rfloor - r} (1/2+\gamma/2)^{\lfloor \frac{k}{2} \rfloor - r} (1/2-\gamma/2)^{\lceil \frac{k}{2} \rceil - i - 1 + r} & \text{if } i - \frac{k}{2} < r \le \frac{k}{2} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } \alpha_{\max}^i = \max_{0 \le r \le i} \alpha_r^i$$

4. choose a real number x uniformly at random from the range $0 \le x \le 1$.
5. If $x < \alpha_r^i / \alpha_{\max}^i$ then accept the example, and return it as the result, else reject it and jump to 1. In each case, update #accept and #reject accordingly.

The abort condition:

$$\texttt{\#accept} + \texttt{\#reject} > \frac{2k\gamma\alpha_{\max}^i}{\epsilon(1-\epsilon)} \max\left(\texttt{\#accept}, 4\ln\frac{16k^2\gamma\alpha_{\max}^i}{\delta\epsilon(1-\epsilon)}\right)$$

Figure 2.4: A description of the algorithm for boosting by filtering.

as *triggering* the abort condition. When the abort is triggered, it stops the execution of procedure **FiltEX** *and* the run of procedure **WeakLearn** that called it, and returns control to statement (2.b). A random hypothesis is then put in place of the hypotheses that was supposed to be generated by **WeakLearn**. A random hypothesis is simply an algorithm that for any $x \in X$ generates a label in $\{0,1\}$ by flipping a fair coin. The abort condition is defined as a function of two counters, `#accept` and `#reject` that are incremented each time an example, generated by **EX** is accepted or rejected respectively. Both counters are reset to zero each time the index i in statement (2) is incremented.

In order to analyze Algorithm $\mathbf{B}_{\mathrm{Filt}}$ we need to go back to the analysis of the underlying majority-vote game. In order to do that we introduce again some of the notation used in Section 2.2 and define it in the context of our new problem.

Let X be the sample space over which a probability distribution $\mathcal{D}$ is defined. Define $\{X_0^i, X_1^i, \ldots, X_i^i\}$ to be a partition of X into $i+1$ sets where X_r^i consists of that set of the sample space that is labeled correctly by r out of the first i hypotheses. Define the following quantities related to this partition:

$M_r^i = X_r^i \cap X_{r+1}^{i+1}$	the subset of X_r^i that is correctly labeled by the $i+1$st hypothesis
$q_r^i = \Pr(X_r^i)$	
$x_r^i = \frac{\Pr(M_r^i)}{\Pr(X_r^i)}$	the probability of a random example to be correctly labeled by h_M given that it is in X_r^i

Finally, denote by t_i the expected value of the weighting factor α_r^i w.r.t. the simulated distribution used in iteration number i, i.e.

$$t_i = \sum_{r=0}^{i} q_r^i \alpha_r^i \tag{2.14}$$

The probability of accepting a random example during the construction of h_{i+1} is $t_i/\alpha_{\max}^i$

We start our analysis by quantifying the reliability of the abort condition. We say that when $t_i < \epsilon(1-\epsilon)/(k\gamma)$, then triggering the abort condition is "justified", the following lemma shows that most triggerings are justified.

Lemma 2.3.5: *For all $0 \le i \le k-1$, the probability, over the distribution of the examples, that an abort is triggered during the generation of h_{i+1}, given that $t_i \ge \epsilon(1-\epsilon)/(k\gamma)$, is smaller than $\delta/2k$.*

Proof: We start by recasting the abort condition in a notation that is more convenient for the analysis. Let $n =$ `#accept` + `#reject` and $m =$ `#reject`. We define the constants $c = \epsilon(1-\epsilon)\ /\ k\gamma\alpha_{\max}^i$ and $n_0 = (8/c)\ln(16k/c\delta)$. Using this notation we say that an abort occurs after testing the nth example if $n > n_0$ and $m < cn/2$. We use $q = t_i/\alpha_{\max}^i$ to denote the probability that **FiltEX** accepts a random example generated by **EX**. Thus the claim that we want to prove is that if $q \ge c$ then the probability of an abort (during any one of the k iterations) is smaller than $\delta/2k$. This probability can be written as a sum of the probabilities of aborting after each example after example number n_0. We can bound the probability of aborting after the nth example using Chernoff bounds as follows:

$$\Pr(m < cn/2) \le e^{-cn/8}\ .$$

Summing this probability over all possible values of n we get that

$$\Pr(\text{abort occurs after } n > n_0 \text{ examples}) < \sum_{n=n_0}^{\infty} e^{-cn/8} = \frac{e^{-cn_0/8}}{1-e^{-c/8}} < \frac{8}{c} e^{-cn_0/8} < \frac{\delta}{2k} ,$$

which proves the claim. ∎

In order for the algorithm $\mathbf{B}_{\text{Filt}}$ to work successfully, we need the reliability of **WeakLearn** to be high. However, as noted by Haussler et. al. [Haussler *et al.*, 1991a], it is easy to boost the reliability of a learning algorithm. We give the performance of one possible reliability-boosting algorithm, $\mathbf{B}_{\text{Rel}}$ in the following lemma. The proof of the lemma and the description of the algorithm are given in Appendix A.1.

Lemma 2.3.6: *Assume* **WeakLearn** *is a learning algorithm that generates hypotheses whose error is smaller than* $1/2-\gamma$ *with probability at least* $\lambda > 0$*, using* m_0 *examples. Then, for any* $\delta > 0$*, Algorithm* $\mathbf{B}_{\text{Rel}}$*, will generate hypotheses whose error is smaller than* $1/2-\gamma/2$ *with probability* $1-\delta$*. Furthermore, the number of examples required by algorithm* $\mathbf{B}_{\text{Rel}}$ *is at most*

$$\frac{8}{\gamma^2}\left(\ln\ln\frac{2}{\delta} + \ln\frac{1}{\delta\lambda}\right) + \frac{m_0}{\lambda}\ln\frac{2}{\delta} .$$

We now give the two main theorems regarding $\mathbf{B}_{\text{Filt}}$. The first theorem proves the correctness of the algorithm and the second proves a bound on the number of training examples required by the algorithm.

Theorem 2.3.7: *If* **WeakLearn** *is a learning algorithm that, for any distribution over the sample space* X *and any* $c \in \mathbf{C}$*, generates a hypothesis whose error is smaller than* $1/2-\gamma$ *with probability* λ *for some* $\gamma, \lambda > 0$*. Then, for any* $\delta, \epsilon > 0$*, the algorithm* $\mathbf{B}_{\text{Filt}}$*, given* ϵ *and* δ*, generates a hypothesis whose error is smaller than* ϵ *with probability at least* $1-\delta$*.*

The proof of this theorem is based on the potential function, β_r^i, defined in the proof of Theorem 2.2.1. From Lemma 2.2.2 we know that the average potential does not increase when the weightor uses the weighting scheme and the chooser plays according to the rule, which corresponds, in the context of learning, to the fact that **WeakLearn** generates a hypothesis with error smaller than $1/2-\gamma$. The following lemma is a refinement of Lemma 2.2.2 that describes the increase in the average potential if the error of the hypothesis is different from $1/2-\gamma$.

Lemma 2.3.8: *If the error of the hypothesis used in the* i*th hypothesis is* $1/2-\hat{\gamma}_i$ *then we have the relationship*

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^i + (\gamma-\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i\alpha_r^i .$$

Proof: Recall Equation (2.10) from the proof of Lemma 2.2.2:

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\left[(1-x_r^i)\beta_r^{i+1} + x_r^i\beta_{r+1}^{i+1}\right] = \sum_{r=0}^{i} q_r^i\beta_r^{i+1} + \sum_{r=0}^{i} q_r^i x_r^i(\beta_{r+1}^{i+1}-\beta_r^{i+1}) .$$

Recall that the weight in the ith iteration of the majority-vote game corresponds to the probability according to the filtered distribution that is observed by **WeakLearn** during the ith iteration of $\mathbf{B}_{\text{Filt}}$. From this, and the definition of $\hat{\gamma}_i$, we get, instead of Equation (2.11), that

$$\sum_{r=0}^{i} q_r^i x_r^i(\beta_{r+1}^{i+1}-\beta_r^{i+1}) = (1/2+\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i(\beta_{r+1}^{i+1}-\beta_r^{i+1}) . \tag{2.15}$$

Combining Equations (2.10) and (2.15) we get:

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^{i+1} \quad + \quad (1/2+\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i(\beta_{r+1}^{i+1}-\beta_r^{i+1})$$

$$= \sum_{r=0}^{i} q_r^i\beta_r^{i+1} \quad + \quad (1/2+\gamma)\sum_{r=0}^{i} q_r^i(\beta_{r+1}^{i+1}-\beta_r^{i+1}) \quad + \quad (\gamma-\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i(\beta_r^{i+1}-\beta_{r+1}^{i+1})$$

$$= \sum_{r=0}^{i} q_r^i\left[(1/2+\gamma)\beta_{r+1}^{i+1}+(1/2-\gamma)\beta_r^{i+1}\right] \quad + \quad (\gamma-\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i(\beta_r^{i+1}-\beta_{r+1}^{i+1}) \ .$$

Using Equation (2.6) for the first term and Equation (2.8) for the second term we find that

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^i \quad + \quad (\gamma-\hat{\gamma}_i)\sum_{r=0}^{i} q_r^i\alpha_r^i \ ,$$

which is the statement of the lemma. ∎

Proof of Theorem 2.3.7 From Lemma 2.3.5 we know that the probability that any of the times the abort condition has been triggered is unjustified is smaller than $\delta/2$. On the other hand, the properties of Algorithm $\mathbf{B}_{\mathrm{Rel}}$, given in Lemma 2.3.6, guarantee that for each iteration, $0 \le i \le k-1$, the probability that the error of h_i is larger than $1/2-\gamma/2$ is smaller than $\delta/2k$. Combining these claims we get that with probability at least $1-\delta$ all the hypotheses have error smaller than $1/2-\gamma/2$ and all the times the abort condition is triggered are justified. We shall now show that in this case the error of h_M is smaller than ϵ.

For all the iterations $1 \le i \le k$ in which the abort condition is not triggered, i.e. the hypothesis h_i is successfully generated, we know from Lemma 2.2.2, that the average potential does not increase. On the other hand, the error of a random coin flip with respect to any distribution over the examples, is, by definition, one half. Thus we get from Lemma 2.3.8 that in the aborted iterations the average potential increases by at most $\gamma\sum_{r=0}^{i} q_r^i\alpha_r^i = \gamma t_i$. As we assume all the aborts are justified, we know that $t_i < \frac{\epsilon(1-\epsilon)}{k\gamma}$. Thus in k iterations the potential increases by at most $\epsilon(1-\epsilon)$. We now follow the same argument as in the proof of Theorem 2.2.1. As the number of iterations, k, is chosen so that $\beta_0^0 \le \epsilon^2$, we get that

$$\Pr\left(h_M(x) \neq c(x)\right) = \sum_{r=0}^{k} q_r^k\beta_r^k \le \beta_0^0 + \epsilon(1-\epsilon) \le \epsilon^2 + \epsilon(1-\epsilon) = \epsilon \ ,$$

where the probability is taken with respect to both the random choice of x according to $\mathcal{D}$, and the random coin flips of the dummy weak hypotheses. ∎

Theorem 2.3.9: *The number of training examples required by* $\mathbf{B}_{\mathrm{Filt}}$ *is smaller than*

$$m = \frac{8\sqrt{2}e^{1/12}}{\sqrt{3\pi}}\frac{k^{3/2}\gamma}{\epsilon(1-\epsilon)}\max\left(m_R, 4\ln\frac{32k^2\gamma}{\delta\epsilon}\right) < \frac{65}{\epsilon\gamma^2}\left(\ln\frac{1}{\epsilon}\right)^{3/2}\max\left(m_R, 12\ln\frac{8\ln 1/\epsilon}{\gamma\delta\epsilon}\right) \ , \qquad (2.16)$$

where k is the number of iterations as chosen in line 1 of $\mathbf{B}_{\mathrm{Filt}}$ *and the inequality is obtained by using the suggested choice of k. The variable m_R denotes the number of examples for generating a weak hypothesis with reliability $1-\delta/2k$ by* $\mathbf{B}_{\mathrm{Rel}}$ *and is equal to:*

$$m_R = \frac{m_0}{\lambda}\ln\frac{4k}{\delta} + \frac{8}{\gamma^2}\left(\ln\ln\frac{4k}{\delta} + \ln\frac{2k}{\delta\lambda}\right) \ .$$

As discussed above the factor $\alpha^i_{\max}$ is chosen so that the probability of accepting a random example is maximized without distorting the simulated distribution. As the value of $\alpha^i_{\max}$ plays a critical role in the proof of the Theorem 2.3.9. we start by presenting a tight upper bound on this value.

Lemma 2.3.10: *For all iterations* $0 \leq i \leq k-2$ *of* $\mathbf{B}_{\text{Filt}}$,

$$\alpha^i_{\max} \leq \sqrt{\frac{8}{3\pi(k-i-1)}} e^{1/12}$$

The proof is given in Appendix A.3.

Proof of Theorem 2.3.9 The number of examples that are required by $\mathbf{B}_{\text{Rel}}$ to generate a hypothesis that has error smaller than $1/2 - \gamma/2$ with probability larger than $1 - \delta/2k$, denoted m_R, is easily bounded using Lemma 2.3.6. The abort condition guarantees that the number of examples that are tested by **FiltEX** during iteration i is at most

$$\frac{2k\gamma\alpha^i_{\max}}{\epsilon(1-\epsilon)} \max\left(m_R, 4\ln\frac{32k^2\gamma}{\delta\epsilon}\right) .$$

Thus the total number of examples is bounded by

$$\max\left(m_R, 4\ln\frac{32k^2\gamma}{\delta\epsilon}\right) \frac{2k\gamma}{\epsilon(1-\epsilon)} \sum_{i=0}^{k-1} \alpha^i_{\max} . \tag{2.17}$$

Using Lemma 2.3.10 for $0 \leq i \leq k-2$ and observing that $\alpha^{k-1}_{\max} = 1$ we can bound the sum by

$$\sum_{i=0}^{k-1} \alpha^i_{\max} \leq \sqrt{\frac{8e^{1/6}}{3\pi}} \left(\sum_{j=1}^{k-1} \frac{1}{\sqrt{j}} + 1\right) < \sqrt{\frac{8e^{1/6}}{3\pi}} 2\sqrt{k} . \tag{2.18}$$

Where the last inequality is true because

$$\sum_{j=1}^{k-1} \frac{1}{\sqrt{j}} < 1 + \int_1^{k-1} \frac{1}{\sqrt{x}} dx = 2\sqrt{k-1} - 1 .$$

Combining 2.17 and 2.18 we get the first inequality in 2.16 and plugging in the choice $k = \frac{4}{\gamma^2}\ln\frac{1}{\epsilon}$ we get the second inequality. ∎

We conclude this section by briefly discussing the time and space complexity of $\mathbf{B}_{\text{Filt}}$. Assuming a uniform bound on the running time of **WeakLearn**, it is clear that the time complexity of $\mathbf{B}_{\text{Filt}}$ is dominated by the time spent in line 3. of **FiltEX** to calculate the labels assigned to the prospective example by the currently available weak hypotheses. As this time is proportional to the number of weak hypotheses available, we get that the time complexity of $\mathbf{B}_{\text{Filt}}$ is at most k times the sample complexity of $\mathbf{B}_{\text{Filt}}$. Similarly, assuming a uniform space complexity on **WeakLearn** and on the size of the hypotheses that it generates, it is clear that the space complexity of $\mathbf{B}_{\text{Filt}}$ is proportional to k, the number of hypotheses that need to be stored in memory.

2.3.4 Randomized learning algorithms and randomized hypotheses

In our discussion so far, we have concentrated on boosting deterministic weak learning algorithms that generate deterministic hypotheses. In this section we show that our results transfer, with little or no change, to the more general case in which both the weak learning algorithm and its hypotheses can be randomized, i.e. make use of flipping random coins.

Note that the data to the learning algorithm and the hypothesis already has a large degree of randomness, as it consists of examples that are chosen at random. We now show a simple transformation that translates randomized learning algorithms into deterministic learning algorithms on a different sample space.

For our analysis we use the convention that the random bits that are used by a randomized algorithm are given to the algorithm as input when it is called. More specifically, we assume the algorithm is given a real valued random number, r, chosen uniformly at random from $[0,1]$ whose binary expansion is used as an infinite source of random bits.[11] We shall take special care that each bit in the binary expansion is used at most once during the run of the algorithm. Thus any random bit used at any point in the algorithm is independent of any other bit. For that reason the distribution of the outcome of the algorithm is equivalent to the distribution generated if each random bit is chosen by an independent coin flip. Thus the transformations we present are only tools for analyzing the sample complexity of the learning algorithm, and the sample and additional computation time of the transformed algorithms that is a result of using this special convention can be ignored.

Assume A is a randomized learning algorithm that generates randomized hypotheses. Assume A can learn the concept class $\mathbf{C}$ for any distribution $\mathcal{D}$ on the sample space X. We now define a mapping μ that maps $X, \mathbf{C}, A$ and $\mathcal{D}$ to $X', \mathbf{C}', A'$ and $\mathcal{D}'$, where A' is a deterministic learning algorithm that generates deterministic hypotheses. The sample space X' consists of pairs of the form $\langle x, r\rangle$, where $x \in X$ and $r \in [0,1)$. The probability measure $\mathcal{D}'$ is the measure generated by the cross product between the distribution $\mathcal{D}$ and the uniform distribution on $[0,1)$. Each concept $c \in \mathbf{C}$ is mapped to a concept $c' \in \mathbf{C}'$ such that for all $\langle x, r\rangle$, $c'(\langle x, r\rangle) = c(x)$. Finally, the algorithm A', receiving the training examples $\{(\langle x_1, r_1\rangle, l_1), \ldots, (\langle x_m, r_m\rangle, l_m)\}$, runs the algorithm A on the sample $\{(x_1, l_1), \ldots, (x_m, l_m)\}$, together with the number r_1, that is used by A as its source of random bits. The hypothesis h, generated by A, is transformed in a similar way, h', upon receiving an instance $\langle x, r\rangle$ as input, calls h to label x, giving it r as its source of random bits.

Note that an infinite sequence of bits can be partitioned into an infinite number of infinite subsequences. For concreteness, we define the nth subsequence of r to consist of the bits whose indices can be written as $(2i-1)2^{n-1}$ for some positive integer i. We denote this subsequence by r_n. Note that if r is chosen uniformly at random then all of its subsequences are also uniformly distributed.

Using these definitions we can now show how boosting the randomized learning algorithm A can be viewed as boosting the deterministic algorithm A' over the larger sample space. Transforming the algorithm for boosting by filtering, $\mathbf{B}_{\text{Filt}}$, is simpler. The change takes place in the procedure **FiltEX**. In

[11] We assume some convention is used for selecting one of the binary expansions when the expansion is not unique.

each iteration the procedure receives an example $\langle x, r \rangle \in X'$ chosen at random according to $\mathcal{D}'$. It then separates x and r, and maps r into $r_1, \ldots, r_{i+1}$, which are independent random bit sequences. Sequences 1 to i are used for calculating $h_1(x, r_1), \ldots, h_i(x, r_i)$. Sequence number $i+1$ is returned to **WeakLearn**, in this case the algorithm A, for use as its source of random bits. Using this transformation the proofs of Theorems 2.3.7 and 2.3.9 can be used without change, and thus $\mathbf{B}_{\text{Filt}}$ works equally well for randomized and deterministic learning algorithms.

The analysis of the algorithm for boosting by sampling, $\mathbf{B}_{\text{Samp}}$, is somewhat more complicated. That is because the same examples are repeatedly fed into A. Since the examples include the source of random bits, this might undesired dependencies between random bits used in different runs of A. To avoid this problem, we assume that an additional integer parameter, which we denote q, is supplied to A. This parameter directs algorithm A to use, as it source of random bits, the qth *subsequence* of the random sequence with which it is supplied. The parameter q is different each time A is called, and thus the random bits used by A are guaranteed to be independent. However, this addition changes somewhat the proof of Theorem 2.3.3, forcing us to increase the size of the sample that is used by $\mathbf{B}_{\text{Samp}}$, as is summarized in the following theorem

Theorem 2.3.11: *Let* **WeakLearn** *be a randomized learning algorithm that generates, with probability $\lambda > 0$ over its internal randomization and the random choice of the training examples, a randomized hypothesis whose error is smaller than $1/2 - \gamma$, for some $\gamma > 0$. Assume the number of training examples required to achieve this is uniformly bounded by m_0. Suppose that m, the size of the sample used by $\mathbf{B}_{\text{Samp}}$, obeys the following inequality:*

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{2}{\delta} + \frac{m_0}{2}\left(\frac{\ln m + 1}{\gamma}\right)^2 + \frac{\ln m}{2\gamma^2}\left(\ln\frac{1}{\lambda} + \ln\ln\frac{1}{\gamma^2\delta} + \ln\ln\ln m\right)\right) .$$

Then with probability at least $1 - \delta$, the hypothesis h_M generated by $\mathbf{B}_{\text{Samp}}$ has error smaller than ϵ.

Proof: The essential difference from the proof of Theorem 2.3.3 is that the number of possible hypotheses that can be generated from the sample is larger. In Theorem 2.3.3 this number is equal to the number of subsequences of size d that can be chosen from a sequence of size m, i.e. m^d. In our case it is the number of subsequences times the number of combinations of values of the parameter q that could have been used in the generation of the k good hypotheses. Assume that $q = ir + l$ where $i = 0 \ldots k-1$ is the number of hypotheses that have been generated so far, $l = 1 \ldots r$ is the counter of the attempts to generate a good ith hypothesis and $r = (1/\lambda)\ln(2k/\delta)$ (These indices are used in statement 3 and 3.a in Figure 2.3). Using this convention it is clear that each one of the hypotheses can be chosen using one of r values, and the total number of combinations of values of q is r^k. Thus the basic inequality that replaces inequality 2.13 is

$$r^k m^d (1-\epsilon)^{m-d} < \delta/2 . \tag{2.19}$$

And solving for m that satisfies this inequality we get the statement of the theorem. ∎

2.3.5 The resources needed for polynomial PAC learning

So far in this paper we have considered learning algorithms that are designed to work for a single fixed concept class defined over a single fixed sample space. However, most learning algorithms can be used for

a *family* of concept classes, and one is then interested in the way the performance of the learning algorithm depends on the complexity of the concept class. Valiant [Valiant, 1984a] presented a framework, called the PAC[12] learning framework, in which such quantification can be done. This framework is one of the most well studied frameworks in computational learning theory. In this section we show the implications of our work on the PAC learning framework.

We start by presenting some notation following Haussler et. al. [Haussler *et al.*, 1991a]. Assume that the sample space is a union of sample spaces of increasing complexity: $X = \cup_{n=1}^{\infty} X_n$. Similarly assume that the concept class that maps points in X_n to $\{0,1\}$ is defined as a union of concept classes of increasing complexity: $\mathbf{C}_n = \cup_{s=1}^{\infty} \mathbf{C}_{n,s}$. The indices n and s usually denote the length of the description of an instance and a concept in some encoding scheme for X and for $\mathbf{C}$ respectively.

We say that a concept class $\mathbf{C}$ is *learnable*, or *strongly learnable*, if there exists a learning algorithm A, and polynomials $p_1(\cdot,\cdot,\cdot,\cdot)$, $p_2(\cdot,\cdot,\cdot,\cdot)$ such that:

- For any n, s and any $\epsilon, \delta > 0$, the algorithm A, given n, s, ϵ, δ and access to an example oracle $\mathbf{EX}$, can learn any concept $c \in \mathbf{C}_{n,s}$ with respect to any distribution $\mathcal{D}$ on X_n, and generate a hypothesis that has error smaller than ϵ with probability larger than $1 - \delta$.
- The sample complexity of A, i.e. the number of calls that A makes to $\mathbf{EX}$, is smaller than $p_1(n, s, 1/\epsilon, 1/\delta)$.
- The running time of A is polynomial in $p_2(n, s, 1/\epsilon, 1/\delta)$.

Kearns and Valiant [Kearns and Valiant, 1988, Kearns and Valiant, 1989] introduced a weaker form of learnability in which the error cannot necessarily be made arbitrarily small. A concept class $\mathbf{C}$ is *weakly learnable* if there exists a learning algorithm A, and polynomials $p_1(\cdot,\cdot,\cdot)$, $p_2(\cdot,\cdot,\cdot)$ and $p_3(\cdot,\cdot)$ such that:

- For any n, s and any $\delta > 0$, the algorithm A, given n, s, δ and access to an example oracle $\mathbf{EX}$, can learn any concept $c \in \mathbf{C}_{n,s}$ with respect to any distribution $\mathcal{D}$ on X_n, and generate a hypothesis that has error smaller than $1/2 - 1/p_3(n, s)$ with probability larger than $1 - \delta$.
- The sample complexity of A, i.e. the number of calls that A makes to $\mathbf{EX}$, is smaller than $p_1(n, s, 1/\delta)$.
- The running time of A is polynomial in $p_2(n, s, 1/\delta)$.

In other words, a weak learning algorithm produces a prediction rule that performs just slightly better than random guessing.

Schapire [Schapire, 1990] has shown that the notions of weak and strong PAC learning are equivalent. Moreover, the boosting algorithm he invented provides an effective way for translating any weak learning algorithm into a strong learning algorithm. The boosting algorithm $\mathbf{B}_{\mathrm{Filt}}$ presented in this paper provides a more efficient translation of weak learning algorithms to strong learning algorithms. A simple application of Theorem 2.3.9 gives the following upper bound on the resources required for PAC learning.

Theorem 2.3.12: *If* $\mathbf{C}$ *is a weakly PAC learnable concept class, parameterized by* n *and* s *in the standard way [Haussler* et al.*, 1991a], then there exists a PAC learning algorithm for* $\mathbf{C}$ *that learns with accuracy* ϵ *and reliability* δ *and:*

- *requires a sample of size*
  $(1/\epsilon)(\log 1/\epsilon)^{3/2}(\log\log 1/\epsilon + \log 1/\delta)p_1(n, s)$,

[12] PAC learning stands for Probably Approximately Correct learning.

- *halts in time*
  $(1/\epsilon)(\log 1/\epsilon)^{5/2}(\log\log 1/\epsilon + \log 1/\delta)p_2(n,s)$,
- *uses space* $(\log 1/\epsilon)(\log\log 1/\epsilon + \log 1/\delta)p_3(n,s)$, *and*
- *outputs hypotheses of size* $(\log 1/\epsilon)p_4(n,s)$ *evaluatable in time* $(\log 1/\epsilon)p_5(n,s)$

for some polynomials p_1, p_2, p_3, p_4 *and* p_5.

Compare this theorem to Theorem 4 in [Schapire, 1990]. The statement there is that the dependence of the sample and time complexity on ϵ is $O(1/\epsilon \text{ poly}(1/\epsilon))$, and that that the other dependencies on $1/\epsilon$ are poly-logarithmic. Our theorem tightens these bounds by giving the explicit powers in the polynomials over $\log(1/\epsilon)$ and $\log(1/\delta)$. Moreover, our more detailed bound, given in Theorem 2.3.9, shows explicitly the dependence on the parameters γ and m_0, which are hidden in the polynomials of the above described theorems. In the next section we show that some of these upper bounds are optimal.

2.3.6 Relations to other bounds

The bounds given in Theorems 2.3.9 and 2.3.12 are currently the best known bounds on the resources required for polynomial PAC learning of an arbitrary PAC learnable class. In this section we relate our results to known lower bounds, and indicate where further improvement might be possible.

Theorem 2.3.12 shows that for any learnable concept class there exists an efficient learning algorithm for which the dependence of the sample size on the required accuracy, when all other parameters are fixed, is $O(1/\epsilon(\log 1/\epsilon)^{3/2})$. A general lower bound of $\Omega(1/\epsilon)$ is given in [Blumer *et al.*, 1986] for learning any "non-trivial" concept class. This lower bound holds without regard to computational constraints on the learning algorithm. There exists a matching upper-bound, given in [Haussler *et al.*, 1988][Theorem 5.1], which says that, ignoring dependence on other parameters, any concept class that can be learned using a sample of size polynomial in $1/\epsilon$ can be learned using a sample of size $O(1/\epsilon)$. The truth might be either that our upper bound can be reduced to match the lower bound, or that there exists a higher lower bound on the sample complexity of learning algorithms that are in RP. However, a result of the second type would be very surprising because it would imply that $RP \neq NP$.

The number of weak hypotheses that are combined by our boosting algorithms is $O(1/\gamma^2 \ln(1/\epsilon))$. We now show that this dependence of the number of required weak hypotheses on ϵ and γ is the best possible for any general boosting algorithm. In order to formalize this claim we have to first define a separation between the weak learner, which does the actual learning, and the boosting algorithm, that can learn only by calling the weak learner.

Assume the examples are given as a sequence of pairs $\langle (x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\rangle$ where $x_i \in X$, $y_i \in \{0,1\}$. The boosting algorithm is a PAC learning algorithm that has direct access only to the y_i part of each example. However, in addition to that, it has access to a learning oracle **WeakLearn** and to a labeling oracle **Label**. The input to **WeakLearn** is a set of indices $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq N$. **WeakLearn** is a learning algorithm and uses as examples the subset $\langle (x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \ldots, (x_{i_k}, y_{i_k})\rangle$ of the sample. It is required to generate a hypothesis $h : X \to \{0,1\}$ that has error $\epsilon_0 = 1/2 - \gamma$ and has reliability $1 - \delta$ if the examples it uses as input are independently drawn from some distribution. The booster can use the

hypotheses returned by **WeakLearn** by calling the second oracle, **Label**, to compute the label given by the hypothesis to any example in the sample, i.e. $\mathbf{Label}(h, i) = h(x_i)$. [13]

Assuming these restrictions, we can apply optimality argument from Section 2.2.1, to give a lower bound on the minimal number of weak hypotheses that have to be combined for boosting. Assume **WeakLearn** is a learning algorithm that generates a randomized hypothesis that is simply the correct concept with independent random noise of $1/2 - \gamma$ applied to the label.

$$h_i(x) = \begin{cases} c(x) & \text{with probability } 1/2 + \gamma \\ 1 - c(x) & \text{with probability } 1/2 - \gamma \end{cases} .$$

Assume the errors of the different hypotheses are independent and that[14] $Pr(c(x) = 0) = Pr(c(x) = 1) = 1/2$. Because of the symmetry in the definitions it is easy to show that in this case the optimal way of combining the outputs of the hypotheses to get the most accurate prediction of $c(x)$ is to take the majority function over all the hypotheses. In this case the number of hypotheses required for achieving accuracy of ϵ is $\frac{1}{2}\gamma^{-2}\log 1/\epsilon$. This shows that the number of weak hypotheses that are combined by $\mathbf{B}_{\mathrm{Filt}}$ is at most eight times the optimum.

As a final comment, we note that as the dependence of the size of the output of a general boosting algorithm on γ is $\Omega(\gamma^{-2})$, the running time of the algorithm necessarily has the same dependence. However, if we fix δ, then the dependence of the running time of $\mathbf{B}_{\mathrm{Filt}}$ on γ is $O(\gamma^{-4})$. The extra factor of γ^{-2} comes from the reliability boosting algorithm, that requires a sample of size $O(\gamma^{-2})$ to guarantee that, with high probability, each weak hypothesis has error smaller than $1/2 - \gamma$. It remains open whether the $O(\gamma^{-4})$ dependence of the running time and of the sample complexity of a general boosting algorithm can be improved.

2.4 Extensions

2.4.1 Using boosting for distribution-specific learning

So far, we have followed the distribution-free paradigm in computational learning and assumed that the learning algorithms that we attempt to boost have complexity bounds that hold uniformly for all input distributions. In this section we show that $\mathbf{B}_{\mathrm{Filt}}$, our second boosting algorithm, can boost learning algorithms whose accuracy is not uniformly bounded for all distributions. We will define a measure of

[13] Restricting a learning algorithm in such a way is natural in the context of hierarchical learning models such as weighted-majority [Littlestone and Warmuth, 1989] and layered neural networks. Some of the analysis of the weighted majority algorithm is concerned with efficiently searching for a good learning algorithm in a pool of algorithms. In this case the learning algorithm has access only to the outputs of the algorithms in the pool and not to the original input. Similarly, in a layered neural network model, units in the deeper hidden layers can receive input only from the layer below them and have no direct access to the input of the network. The process of filtering or sub-sampling can be interpreted, in this context, as a feedback mechanism by which a learning unit higher in the hierarchy directs lower level inputs to concentrate on those examples which will contribute most to the performance of the network as a whole.

[14] If $Pr(c(x) = 0) \neq Pr(c(x) = 1)$ then some decrease in the output hypothesis size is possible. However, the $\Omega(\gamma^{-2})$ dependence is unavoidable.

discrepancy between distributions and show that the accuracy of **WeakLearn** can be allowed to degrade as the discrepancy increases between the filtered distribution that is fed into **WeakLearn** and the distribution that governs the example oracle **EX**. We shall refer to the distribution governing **EX** as the "target" distribution.

From Lemma 2.3.8 we know that the increase in the average potential in the ith iteration is equal to

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^i + (\gamma - \hat{\gamma}_i)\sum_{r=0}^{i} q_r^i\alpha_r^i \ .$$

Where $\hat{\gamma}_i$ is the difference between $1/2$ and the error of h_i with respect to the filtered distribution in the ith iteration. We recall the notation defined in Section 2.3.3: $t_i = \sum_{r=0}^{i} q_r^i\alpha_r^i$ and re-write the last equation

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^i + (\gamma - \hat{\gamma}_i)t_i \ .$$

Recall that the probability of accepting a random example that is tested during the ith iteration is $t_i/\alpha_{\max}^i$. Thus, if the probability of accepting a random example during the ith iteration is small, then the sensitivity of the final accuracy to the accuracy of the ith hypothesis is small. We have already used this fact in the proof of Theorem 2.3.7. There we used it to show that if the probability of accepting a random example is small enough, then a random coin flip can be used instead of the weak hypothesis. In this section we use the same property to relax the requirements on the accuracy of the hypotheses generated by **WeakLearn** for distributions that are far from the target distribution.

The following lemma shows how the requirements on the accuracy of the hypotheses generated by **WeakLearn** can be relaxed, allowing the generation of hypotheses whose error is larger than $1/2 - \gamma$.

Lemma 2.4.1: *Let $0 < \gamma, \epsilon \leq 1/2$ be the accuracy parameters supplied to $\mathbf{B}_{\mathrm{Filt}}$, and k be the number of iterations chosen by $\mathbf{B}_{\mathrm{Filt}}$. Let t_i denote $\sum_{r=0}^{i} q_r^i\alpha_r^i$ and let $1/2 - \hat{\gamma}_i$ denote the error of h_i with respect to the filtered distribution in the ith iteration.*

If, for each iteration $0 \leq i \leq k-1$ we have

$$\hat{\gamma}_i \geq \gamma\left(1 - \frac{\epsilon(1-\epsilon)}{t_i\gamma k}\right) \ , \tag{2.20}$$

then the error of h_M, the hypothesis output by $\mathbf{B}_{\mathrm{Filt}}$, with respect to the target distribution, is smaller than ϵ

Proof: From Lemma 2.3.8 we immediately get that the increase of the average potential in each iteration is at most $\epsilon(1-\epsilon)/k$. Thus the total increase in the average potential in all k iterations is $\epsilon(1-\epsilon)$. The rest of the proof follows the same line of argument as the one used for the aborted iterations in the proof of Theorem 2.3.7. ∎

To illustrate the significance of this result, assume that **WeakLearn** generates a hypothesis whose error is $1/2 - \gamma$ when given examples from the target distribution. Our goal is to achieve a higher degree of accuracy on the target distribution by making use of the performance of **WeakLearn** on other distributions. As we know from the main results of this paper, if **WeakLearn** is capable of generating a hypothesis with error smaller than $1/2 - \gamma$ for *any* distribution then boosting can achieve any desired accuracy on the target

distribution. However, using Lemma 2.4.1 boosting can be used even in cases where the accuracy of the hypotheses generated by **WeakLearn** decreases as the distributions supplied to it become more and more different from the target distribution. The slower the decrease in accuracy, the higher the quality that can be achieved by boosting.

We start by simplifying Equation (2.20). By choosing $k = (4/\gamma^2)\ln(1/\epsilon)$, we get an upper bound on the error of h_i as a function of γ and t_i:

$$\hat{\gamma}_i \geq \gamma \left(1 - \frac{\gamma\epsilon(1-\epsilon)}{4t_i \ln(1/\epsilon)}\right) \ .$$

Different choices for γ generate different lower bounds on $\hat{\gamma}_i$ as a function of t_i. An illustration of these lower bounds is given in Figure 2.5.

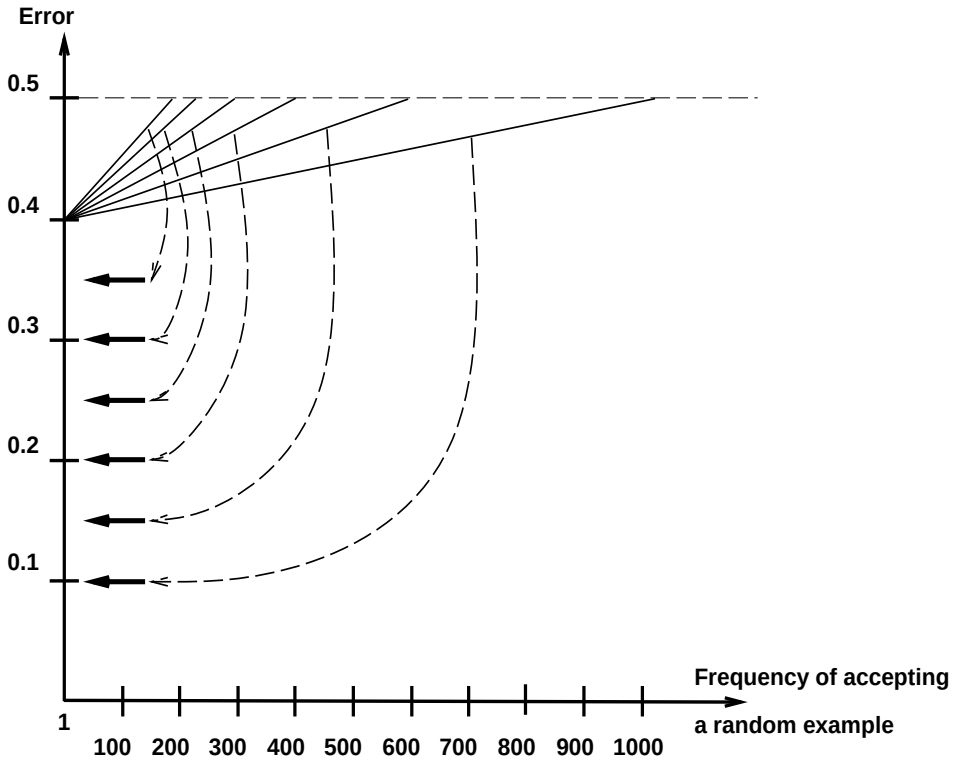


Figure 2.5: **The accuracy that can be achieved using boosting a learner whose accuracy depends on the distribution.** The horizontal line denotes $1/t$, or the number of examples that have to be filtered per accepted example. The origin denotes an acceptance rate of 1, i.e. every example is accepted, which means that the weak learner is observing the original distribution. The horizontal axis denotes the error of the hypotheses. Each sloped line denotes a requirement on the maximal error of the weak learner as a function of the divergence from the target distribution. Each such bound guarantees a different accuracy of the final hypothesis, which is described by the bold arrow on the error axis.

In order to separate the requirements for **WeakLearn** from the particulars of our boosting algorithm, we need to upper bound the value of t_i using a measure of the discrepancy between the target distribution

and the filtered distribution. We shall now define such a measure of discrepancy, show that this measure is closely related to the Kullback-Leibler divergence, and give a stronger version of Theorem 2.3.7 based on this measure.

Definition 1: *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions defined over the same space X and sigma-algebra Σ. The maximal-ratio divergence between $\mathcal{Q}$ and $\mathcal{P}$, denoted $D_M(\mathcal{Q}||\mathcal{P})$, is defined to be*

$$D_M(\mathcal{Q}||\mathcal{P}) \doteq \ln\left(\sup_{A\in\Sigma,\ \mathcal{P}(A)>0}\frac{\mathcal{Q}(A)}{\mathcal{P}(A)}\right) .$$

We now lower bound the maximal ratio divergence using the well-known Kullback-Leibler divergence.

Lemma 2.4.2: *For any two distributions $\mathcal{Q}$ and $\mathcal{P}$, defined on the same measure space,*

$$D_M(\mathcal{Q}||\mathcal{P}) \geq D_{KL}(\mathcal{Q}||\mathcal{P}) .$$

Where $D_{KL}(\mathcal{Q}||\mathcal{P})$ is the Kullback-Leibler divergence, which is defined as

$$D_{KL}(\mathcal{Q}||\mathcal{P}) \doteq E_{x\in\mathcal{Q}}\left(\ln\frac{\mathcal{Q}(x)}{\mathcal{P}(x)}\right) .$$

Proof: If $E_{\mathcal{Q}}\left(\ln\frac{\mathcal{Q}(x)}{\mathcal{P}(x)}\right) \geq a$ then there exists a set A such that $\mathcal{Q}(A) > 0$ and $\ln\frac{\mathcal{Q}(A)}{\mathcal{P}(A)} > a$, which implies that $D_M(\mathcal{Q}||\mathcal{P}) \geq a$. ∎

Note that there is no similar inequality relating the two measures of divergence in the other way. That is because there might be a set A such that $Q(A)$ is very small, so that the contribution of this set to $D_{KL}(\mathcal{Q}||\mathcal{P})$ is negligible, but on the other hand $\mathcal{Q}(A)/\mathcal{P}(A)$ is extremely large.

Using these measures of divergence, we can lower-bound t_i by functions of the divergence between the target distribution and the ith filtered distribution:

Lemma 2.4.3: *If $\mathcal{D}$ is the target distribution, and F_i is the distribution generated by* **FiltEX** *during the ith iteration, then*

$$t_i \leq e^{-D_M(F_i||V)} \leq e^{-D_{KL}(F_i||V)} .$$

Proof: The second inequality follows from Lemma 2.4.2. To prove the first inequality, assume that $D_M(F_i||\mathcal{D}) > a$. Then there exists a set $A \in \Sigma$ such that $\frac{F_i(A)}{\mathcal{D}(A)} > e^a$. Using the definition of the measure generated by filtering in Equation (2.3) we get

$$e^a < \frac{\sum_{r=0}^{i}\mathcal{D}(A\cap X_r^i)\alpha_r^i \ / \ Z_i}{\sum_{r=0}^{i}\mathcal{D}(A\cap X_r^i)} \leq \frac{\sum_{r=0}^{i}\mathcal{D}(A\cap X_r^i) \ / \ Z_i}{\sum_{r=0}^{i}\mathcal{D}(A\cap X_r^i)} = \frac{1}{Z_i} .$$

The inequality holds because $\alpha_r^i \leq 1$ always.[15] Here $Z_i = \sum_{r=0}^{i}\mathcal{D}(X_r^i)\alpha_r^i = t_i$, from which we get $1/t_i \geq e^a$, which proves the lemma. ∎

We now combine the results of Lemmas 2.4.1, 2.4.2 and 2.4.3 to arrive at the following stronger version of Theorem 2.3.7

[15] Notice that a tighter bound can be proved using the bound on $\alpha_{\max}^i = \max_{0\leq i\leq r}\alpha_r^i$ given in Lemma 2.3.10. However, here we avoid using this tighter bound because we want the bound to be independent of i.

Theorem 2.4.4: *Fix a target distribution $\mathcal{D}$ and real valued parameters $\gamma, \epsilon, \delta > 0$.*
If **WeakLearn** *is a learning algorithm that for any distribution $\mathcal{P}$ over the sample space X and any $c \in \mathbf{C}$, generates a hypothesis whose error, w.r.t. $\mathcal{P}$, is smaller than*

$$\frac{1}{2} - \gamma \left(1 - \frac{\epsilon(1-\epsilon)\gamma}{4\ln(1/\epsilon)} e^{D_{KL}(\mathcal{P}||\mathcal{D})}\right) ,$$

then, with probability at least $1-\delta$, the algorithm $\mathbf{B}_{\text{Filt}}$, given the parameters, generates a hypothesis whose error, w.r.t. $\mathcal{D}$, is smaller than ϵ.

Proof: The algorithm uses $k = (4/\gamma^2)\ln(1/\epsilon)$ as given in statement 1. of Algorithm $\mathbf{B}_{\text{Filt}}$ (Figure 2.4). From Lemma 2.4.1 we get that it is enough if the error in the ith iteration is smaller than

$$\frac{1}{2} - \gamma \left(1 - \frac{\epsilon(1-\epsilon)\gamma}{4 t_i \ln(1/\epsilon)}\right) .$$

Combining Lemmas 2.4.3 and 2.4.2, we get that $t_i \leq e^{-D_{KL}(\mathcal{D}||\mathcal{T})}$, which proves the theorem. ∎

Notice that Theorem 2.4.4 assumes that the weak learner is completely reliable, i.e. that it has probability 1 of generating a hypothesis with the desired accuracy. The algorithm can be used for less reliable weak learning algorithms, but there is a subtle point that needs to be addressed in that case. The point is that the number of examples required by $\mathbf{B}_{\text{Rel}}$ in order to increase the reliability is $\Omega(1/\gamma^2)$. Thus if the error of the hypothesis has to be just very slightly smaller than 1/2, the number of examples that are required to test if the hypothesis is good increases without bounds. To avoid this problem the required error has to be set to a smaller value, thus making the detection of a good hypothesis easier. We omit the details of this variant of the boosting algorithm.

2.4.2 Boosting multiple valued concepts

As was noted by Schapire [Schapire, 1991], the generalization of the equivalence between strong and weak learning to concepts with more than two labels does not enjoy the same tightness as the two label case. In the two label case an ability to predict the label with accuracy that is a polynomial fraction better than random guessing is equivalent to strong learning. In the j-label case the probability that a random guess is correct is equal to $1/j$, while the minimal requirement for weak learning to be equivalent to strong learning is to predict correctly with a probability slightly better than a *half*. [16] As any j-valued decision rule can be replaced by $j-1$ binary decision rules of the type: "is the label equal i", the binary boosting algorithm can be used $j-1$ times to generate the desired hypothesis. However, it is possible to perform the boosting process in one pass, generating a simple j-valued hypothesis and eliminating the dependence of the complexity on j. The combination rule that is used is simply the j-valued majority, i.e. the strong hypothesis labels the input with the label given by the largest number of weak hypotheses. The algorithm and its analysis are almost exactly the same as in the binary case, the only difference is that the definition of the filtering factor is based on one more parameter, denoted by t, that is the number of

[16] To realize this, consider a 3-label concept such that for any example there are only *two* possible labels (over the whole concept class). In this case, using a random coin flip to choose one of the two possible labels will give a correct answer half of the time, but the concept class might still be unlearnable [Schapire, 1991].

incorrect hypothesis whose output is not equal to the incorrect label with the largest number of votes. For example, suppose the labels are the ten digits, assume the correct label for some example is "0" and the incorrect label that got the largest number of votes is "9" (irrespective of whether the number of votes "9" got is larger than the number of votes "0" got) then t is the number of votes that the digits "1" to "8" got. The change in Formula (2.1) is that k is replaced by $k-t$:

$$\alpha_{r,t}^{i} = \begin{cases} 0 & \text{if } r \leq i - \frac{k-t}{2} \\ \binom{k-t-i-1}{\lfloor \frac{k-t}{2} \rfloor - r} (\frac{1}{2}+\gamma)^{\lfloor \frac{k-t}{2} \rfloor - r} (\frac{1}{2}-\gamma)^{\lceil \frac{k-t}{2} \rceil - i - 1 + r} & \text{if } i - \frac{k-t}{2} < r \leq \frac{k-t}{2} \\ 0 & \text{if } r > \frac{k-t}{2} \end{cases} \tag{2.21}$$

It is interesting to note that the resources required are completely independent of j, the number of possible labels. This is even true if j is different for different n and s, or if j is infinite, even uncountable! However, the requirement of weak learning for concepts with uncountable ranges is unreasonably hard. The hypothesis must generate the *exact* correct output for more than half the inputs (in probability). In this case the result described in the next section might be more relevant.

2.4.3 Boosting real valued concepts

A modification of the boosting algorithm can be used for boosting learning algorithms for concept classes whose range is a real number (for a review of algorithms for learning real valued functions, see Chapter 5 in [Natarajan, 1991]). This variant of the boosting algorithm transforms learning algorithms that generate hypotheses whose expected error, with respect to the input distribution, is small to algorithms that generate hypotheses whose error is small for most of the input domain.

Assume **C** is a set of functions from R to R and **WeakLearn** is a learning algorithm for **C** . Let p be any density function over R, and let $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$ be a set of examples drawn independently at random according to p and labeled according to some $f \in \mathbf{C}$. Then A, upon observing this sample, generates a hypothesis function g such that with probability larger than $1-\delta$

$$\int_{-\infty}^{+\infty} |f(x) - g(x)| dp(x) < d \ . \tag{2.22}$$

We shall sketch how the boosting algorithm can be used to generate a function h such that with high probability

$$P_p \left(|f(x) - h(x)| > \frac{d}{1/2 - \gamma} \right) < \epsilon \ .$$

Where P_p is the probability according to the density p and $\gamma, \epsilon > 0$ are polynomial fractions.

Using the Markov inequality and setting $\Delta = \frac{d}{1/2-\gamma}$ we get, from Equation (2.22), that

$$P_p \left(|f(x) - g(x)| > \Delta \right) < \frac{1}{2} - \gamma \ .$$

We extend the notion of *agreement* between a concept and a hypothesis on an example x to concepts defined on the reals by saying that f and g "Δ-agree" on x if $|f(x) - g(x)| < \Delta$. Using the extended definition of agreement we can say that **WeakLearn** is a weak-learner for the concept class **C** . If we replace all the places in the boosting algorithm in which it refers to "agree" or "correct" by corresponding references to "Δ-agree" or "Δ-agrees with the true function", we get a boosting algorithm for real valued functions.

Suppose, for simplicity, that we are using algorithm $\mathbf{B}_{\text{Samp}}$. Then the result of running the boosting algorithm over the weak learning algorithm are k real valued functions $h_1(x), \ldots, h_k(x)$ such that for any point in the sample more than $k/2$ of the functions are within δ of the correct value. It is interesting to observe that the results of Theorems 2.3.3 and 2.3.11 hold without change for the real valued case. Thus, by choosing the size of the sample large enough, we are guaranteed that, with probability at least $1-\delta$, more than half of the hypotheses are Δ-correct on all but ϵ of the points of the *whole domain*.

Observe that if more than half of the functions Δ-agree with f on a point x then the median of the functions Δ-agrees with f. From this we get that the median is the natural generalization of the majority for this case. By taking the median of the k weak hypotheses we get:

$$P_p(\text{Median}(h_1, h_2, \ldots, h_k)\ \Delta\text{-agrees with } f) > 1-\epsilon\ .$$

2.4.4 Parallelizing PAC learning

The fact that the boosting by filtering algorithm, $\mathbf{B}_{\text{Filt}}$, accepts only a small fraction of the examples with which it is presented has an interesting implication on the possibility of achieving optimal speed-up when parallelizing learning algorithms.

Observe that the time complexity of $\mathbf{B}_{\text{Filt}}$ is dominated by the time that is spent by the procedure **FiltEX** on checking examples that are eventually rejected. Observe also the probability that any given example is accepted during the generation of the ith hypothesis is constant. In other words, it is independent of whether or not any other example is tested or accepted during the ith stage.

Assume now that we use one of the standard parallel-computation paradigms, such as the PRAM model, and that we have a computer with a p processors at our disposal. Then we can parallelize the procedure **FiltEX** in the following way. Each of the p processors runs the procedure **FiltEX** independently, each making separate calls to **EX**, so that they test different random examples.[17] When one of the p processors accepts an example, all the other processors are halted and their results are ignored.[18] The accepted example is then returned to **WeakLearn** as usual. Recall that out of the $O(1/\epsilon(\ln 1/\epsilon)^{3/2})$ examples that are needed for learning, only $O(\ln 1/\epsilon)$ examples have to be accepted and returned to **WeakLearn**. If the number of processors is $O(1/\epsilon\sqrt{\ln 1/\epsilon})$ then the search for an acceptable example takes expected constant time, so that the expected running time of the boosting algorithm becomes $O(\ln 1/\epsilon)$. If p is smaller, then a p-fold speedup over the serial execution is achieved. We summarize this observation in the following theorem.

Theorem 2.4.5: *If* $\mathbf{C}$ *is a polynomially PAC-learnable concept class then there exists a parallel learning algorithm for* $\mathbf{C}$ *that runs on a PRAM machine with* $O(1/\epsilon)$ *processors whose time complexity dependence on the accuracy is* $O(\log 1/\epsilon)$.

[17] We either assume that the running time of **EX** is negligible or that **EX** can generate many examples at the same time.

[18] We assume that halting all processors can be done in unit time.

2.5 Summary and open problems

The algorithms we have described in this paper give the best upper bounds currently known on the resources required for polynomial PAC learning. While these bounds are in some respects close to optimal, further improvement might still be possible in the dependence of the sample and time complexity on the parameters ϵ and γ.

One undesired property of our boosting algorithm is that it requires prior knowledge of a distribution-independent bound on the accuracy of the hypotheses that **WeakLearn** generates. While guessing a bound is a theoretically feasible solution, it is expensive in practical applications [Drucker, 1992 1993]. Schapire's algorithm is somewhat better in that respect, because if sample complexity is ignored it can be used without having prior knowledge of such a bound, and achieve an improvement over the performance of **WeakLearn** if such a uniform bound exists.

A deeper problem is that the assumption of distribution-independent bounds for learning algorithms often seems to be unreasonable. Theorem 2.4.4 is encouraging in this respect because it shows that boosting can be achieved even without uniform bounds. This might be a sign that a richer, and maybe more realistic theory of learning can be developed in which performance bounds are distribution dependent.

In this paper we have shown that the boosting algorithm can be generalized to multiple-valued concept classes as well as real valued concept classes. However, the results regarding real-valued concept classes are still rather weak, and one would hope that stronger types of boosting can be achieved in that context. The use of boosting in the context of p-concepts [Kearns and Schapire, 1990] is another long standing open problem. Some progress on the problem of boosting in the context of independent label noise has been achieved in a recent work by Aslam and Decatur about boosting learning algorithms in the the statistical query model introduced by Kearns [Kearns, 1993].

Last but not least, boosting has been successfully applied to some practical machine learning problems [Drucker *et al.*, 1993]. Further experimentation with boosting methods will hopefully achieve even better results. Such work will also be useful in pointing to directions in theoretical research that might have a large impact on the practice of machine learning.

2.6 Summary of notation

2.6.1 Concept Learning Notation

The sample space is denoted X, the concept class is denoted $\mathbf{C}$, and the class of hypotheses is denoted H. Typical elements of these spaces are denoted x, c and h respectively. The distribution over X, according to which examples are generated, is denoted by $\mathcal{D}$. We denote by $S = \{(x_1, c(x_1)), \ldots, (x_m, c(x_m))\}$ a sample of m examples, labeled according to $c \in \mathbf{C}$. The accuracy parameter is denoted ϵ, and the reliability parameter is denoted δ. The sample, time, and space required for the learning algorithm under discussion to achieve accuracy ϵ with reliability δ are denoted $m(\epsilon, \delta)$,$s(\epsilon, \delta)$ and $t(\epsilon, \delta)$ respectively.

2.6.2 Notation for the describing boosting

We denote a generic weak learning algorithm by **WeakLearn**. We use ϵ_0 and δ_0 to denote the accuracy and the reliability of **WeakLearn**. Usually ϵ_0 is close to $1/2$ (the accuracy of a random guess) and δ_0 is close to 1 (probability zero of generating an ϵ_0-accurate hypothesis). For this reason we define $\gamma = 1/2 - \epsilon_0$ and $\lambda = 1 - \delta_0$. The number of examples, time and space required by the weak learner to achieve its fixed goals are denoted m_0, t_0, and s_0 respectively. We denote the hypothesis generated by the boosting algorithm by h_M, and the set of all such hypotheses by H_M.

2.6.3 Meaning of common notation in different sections

symbol	Meaning in Majority-Vote Game	Meaning in analysis of $\mathbf{B}_{\text{Samp}}$	Meaning in analysis of $\mathbf{B}_{\text{Filt}}$
k	The total number of iterations in the game.	The total number of weak hypotheses combined by the boosting algorithm.	
$i = 0 \ldots k$	The number of iterations played so far.	The number of weak hypotheses generated so far.	
$r = 0 \ldots i$	The number of marks.	The number of weak hypotheses that are correct	
X_r^i	The points that have been marked r times in the first i iterations	The points in the sample on which r out of the first i weak hypotheses are correct	The points in X on which r out of the first i weak hypotheses are correct
$V(A)$	The value of the set A	The number of sample points in A	The probability of A according to the distribution $\mathcal{D}$
$W_i(A)$	The weight of the set A in the ith iteration	The sum of the weights assigned to the sample points in A using hypotheses $h_1, \ldots, h_{i-1}$	The probability of A according to the distribution filtered using hypotheses $h_1, \ldots, h_{i-1}$
α_r^i	The weight assigned to points in X_r^i defined in Equation 2.1		$\alpha_i^r / \alpha_{\max}^i$ is the probability of accepting an example from X_r^i during the ith iteration. where $\alpha_{\max}^i = \max_{0 \le r \le i} \alpha_r^i$
β_r^i	The potential of the points in X_r^i, defined in Equation 2.7		
$q_r^i = V(X_r^i)$	The value of X_r^i	The number of sample points in X_r^i	The probability of X_r^i according to the distribution $\mathcal{D}$
$x_r^i = \frac{V(X_r^i \cap X_{r+1}^{i+1})}{V(X_r^i)}$	The fraction, in terms of value) of X_r^i that is marked in the ith iteration	The fraction of the points of X_r^i on which h_{i+1} is correct	The fraction (in terms of the distribution $\mathcal{D}$) of X_r^i on which h_{i+1} is correct
L The loss set	The set of points marked less than $k/2$ times in the k iterations	The sample points on which the majority vote is incorrect i.e. the empty set	The set of points in X on which the majority vote is incorrect, i.e. the set of points on which h_M is incorrect.

2.6.4 Special Notation

- $t_i = \sum_{r=0}^{i} q_r^i \alpha_r^i$
  The expected weight of a random example in the ith iteration. This notation is used in the analysis of $\mathbf{B}_{\text{Filt}}$ because $t_i / \alpha_{\max}^i$ is the probability of accepting a random example during the ith iteration.
- $\hat{\gamma}_i$ - the actual edge of the ith weak hypothesis, h_i. In other words, the error of h_i, with respect to the ith filtered distribution, is $1/2 - \hat{\gamma}_i$.

- m_R - Denotes the number of examples required by $\mathbf{B}_{\mathrm{Rel}}$ for generating a weak hypothesis with the desired reliability.

3. Accelerating learning using Query by Committee

3.1 Introduction

Most of the research on the theory of learning from random examples is based on a paradigm in which the learner is both trained and tested on examples drawn at random from the same distribution. In this paradigm the learner is passive and has no control over the information that it receives. In contrast, in the *query* paradigm, the learner is given the power to ask questions. What does the learner gain from this additional power?

Study of the use of queries in learning [Valiant, 1984b, Angluin, 1988a], has mostly concentrated on algorithms for *exact identification* of the target concept. This type of analysis concentrates on the worst case behavior of the algorithm, and no probabilistic assumptions are made. In contrast, we are interested in algorithms that achieve approximate identification of the target, and our analysis is based on probabilistic assumptions. We assume that both the examples and the target concept are chosen randomly. In particular, we show that queries can help *accelerate* learning of concept classes that are already learnable from just unlabeled data.

This question was previously studied by Eisenberg and Rivest [Eisenberg and Rivest, 1990] in the PAC learning framework. They give a negative result, and show that, for a natural set of concept classes, which they call "dense in themselves", queries are essentially useless. They show that giving the learner the ability to ask membership queries (questions of the type "what is the label of the point x?") in this context does not enable the learner to significantly reduce the total number of labeled examples it needs to observe. The reason is that if the learner observes only a small number of examples, *either passively or actively*, then it can not be sensitive to slight changes in the target concept and in the underlying distribution. An adversary can alter the distribution and the target in a way that will not cause the learner to change its hypothesis, but will increase the error of this hypothesis in a significant way. In this paper we show how some concept classes that are dense in themselves can be learned efficiently if we allow the learner access to random *unlabeled* examples. This added capability enables the learner to maintain its sensitivity to the input distribution, while reducing the number of labels that it needs to know.

Baum [Baum, 1991], proposed a learning algorithm that uses membership queries to avoid the intractability of learning neural networks with hidden units. His algorithm is proved to work for networks with at most 4 hidden units, and there is experimental evidence [Baum and Lang, 1991] that it works for larger networks. However, when Baum and Lang tried to use this algorithm to train a network for classifying handwritten characters, they encountered an unexpected problem [Baum and Lang, 1992]. The problem was that many of the images generated by the algorithm as queries did not contain any recognizable character, they were artificial combinations of character images that had no natural meaning. The learning algorithm that is analyzed in this paper uses random unlabeled instances as queries and in this way avoids the problem encountered by Baum's algorithm.

Our work is derived within the *query filtering* paradigm. In this paradigm, proposed by [Cohn *et al.*, 1990], the learner is given access to a stream of inputs drawn at random from the input distribution. The

learner sees every input, but chooses whether or not to query the teacher for the label. Giving the learner easy access to unlabeled random examples is a very reasonable assumption in many real-life contexts. In applications such as speech recognition, it is often the case that collecting unlabeled data is a highly automatic process, while finding the correct labeling of the data requires expensive human work. Our algorithm uses all of the unlabeled examples and in this way it overcomes the problems pointed out by Rivest and Eisenberg. Learning becomes an interactive process, rather then requesting the human to label all the examples in advance, we let the computer choose the examples whose labels are most informative. Initially, most examples will be informative for the learner, but as the process continues, the prediction capabilities of the learner improve, and it discards most of the examples as non-informative, thus saving the human teacher a large amount of work.

In [Cohn *et al.*, 1990] there are several suggestions for query filters together with some empirical tests of their performance on simple problems. Seung et al.[Seung *et al.*, 1992] have suggested a filter called "query by committee," and analytically calculated its performance for some perceptron-type learning problems. For these problems, they found that the prediction error decreases exponentially fast in the number of queries. In this work we present a more complete and general analysis of query by committee, and show that such an exponential decrease is guaranteed for a general class of learning problems.

The problem of selecting the optimal examples for learning is closely related to the problem of experimental design in statistics (see e.g. [Fedorov, 1972, Atkinson and Donev, 1992]). Experimental design is the analysis of methods for selecting sets of experiments, which correspond to membership queries in the context of learning theory. The goal of a good design is to select experiments in a way that their outcomes, which correspond to labels, give sufficient information for constructing a hypothesis that maximizes some criterion of accuracy. One natural criterion is the accuracy with which the parameters that define the hypothesis can be estimated [Lindley, 1956]. In the context of Bayesian estimation a very general measure of the quality of a query is the reduction in the probability of the set of possible hypotheses that is induced by the answer to the query. Similar suggestions have been made in the perceptron learning literature[Kinzel and Ruján, 1990]. A different experimental design criterion is the accuracy with which the outcome of future experiments, chosen from some constrained domain, can be predicted using the hypothesis. This criterion is very similar to criteria used in learning theory. Both criteria are important for us in this paper. We show that while in the general case the two are not necessarily related, they are related in the case of the query by committee algorithm. Using this relation we prove the efficiency of the algorithm for some specific concept classes.

The paper is organized as follows. In Section 3.2 we present the Bayesian framework of learning in which we analyze our algorithm. In Section 3.3 we present some simple learning problems and demonstrate a case in which the information gain of a query is not the relevant criterion when we are interested in prediction quality. In Section 3.4 we describe the query-by-committee algorithm. In Section 3.5 we prove that there is a close relation between information gain and prediction error for **QBC**. Using this relation we show in Section 3.6 that the prediction error decreases exponentially fast with the number of queries for some natural learning problems. In Section 3.7 we give a broader view on using unlabeled examples for accelerating learning, and in Section3.8 we summarize and point to some potential future directions.

3.2 Preliminaries

We work in a Bayesian model of concept learning [Haussler *et al.*, 1991b]. As in the PAC model, we denote by X an arbitrary sample space over which a distribution $\mathcal{D}$ is defined. In this paper we concentrate on the case where X is a Euclidean space R^d. Each concept is a mapping $c : X \to \{0,1\}$ and a concept class $\mathbf{C}$ is a set of concepts. The Bayesian model differs from the PAC model in that we assume that the target concept is chosen according to a *prior distribution* $\mathcal{P}$ over $\mathbf{C}$ and that this distribution is known to the learner. We shall use the notation $\Pr_{x \in \mathcal{D}}(\cdot)$ to denote the probability of an event when x is chosen at random from X according to $\mathcal{D}$.

We assume that the learning algorithm has access to two oracles: **Sample** and **Label**. A call to **Sample** returns an unlabeled example $x \in X$, chosen according to the (unknown) distribution $\mathcal{D}$. A call to **Label** with input x, returns $c(x)$, the label of x according to the target concept. After making some calls to the two oracles, the learning algorithm is required to output a hypothesis $h : X \to \{0,1\}$. We define the expected error of the learning algorithm as the probability that $h(x) \neq c(x)$, where the probability is taken with respect to the distribution $\mathcal{D}$ over the choice of x, the distribution $\mathcal{P}$ over the choice of c and any random choices made as part of the learning algorithm or of the calculation of the hypothesis h. We shall usually denote the number of calls that the algorithm makes to **Sample** by m and the number of calls to **Label** by n. Our goal is to give algorithms that achieve accuracy ϵ after making $O(1/\epsilon)$ calls to **Sample** and $O(\log 1/\epsilon)$ calls to **Label**.

In our analysis we find it most convenient to view the finite number of examples that the learning algorithm makes to label as being an initial segment of an infinite sequence of examples, all drawn independently at random according to $\mathcal{D}$. We shall denote such a sequence of unlabeled examples by $\vec{X} = \{x_1, x_2 \ldots\}$, and use $\langle \vec{X}, c(\vec{X}) \rangle = \{\langle x_1, c(x_1) \rangle, \langle x_2, c(x_2) \rangle \ldots\}$ to denote the sequence of labeled examples that is generated by applying c to each $x \in \vec{X}$. We use $\vec{X}_{1\ldots m}$ to denote the sequence of the first m elements in $\vec{X}$. We use the terminology of Mitchell [Mitchell, 1978], and define the *version space* generated by the sequence of labeled examples $\langle \vec{X}_{1\ldots m}, c(\vec{X}_{1\ldots m}) \rangle$ to be the set of concepts $c' \in \mathbf{C}$ that are consistent with c on $\vec{X}$, i.e. that $c'(x_i) = c(x_i)$ for all $1 \leq i \leq m$. We denote the version space that corresponds to the first i labeled examples by $V_i = V(\langle \vec{X}_{1\ldots i}, c(\vec{X}_{1\ldots i}) \rangle)$. The initial version space, $V_0 = V(\emptyset)$, is equal to $\mathbf{C}$. The version space is a representation of the information contained in the set of labeled examples observed by the learning algorithm. A natural measure of the progress of the learning process is the rate at which the size of the version space decreases. The *instantaneous information gain* from the ith labeled example in a particular sequence of examples is defined to be $-\log \Pr_{\mathcal{P}}(V_i)/\Pr_{\mathcal{P}}(V_{i-1})$. Summing the instantaneous information gains over a complete sequence of examples we get the *cumulative information gain*, which is defined as

$$\mathcal{I}(\langle x_1, c(x_1) \rangle, \ldots, \langle x_m, c(x_m) \rangle) \doteq -\sum_{i=1}^{m} \log \frac{\Pr_{\mathcal{P}}(V_i)}{\Pr_{\mathcal{P}}(V_{i-1})} = -\log \Pr_{\mathcal{P}}(V_m) \ . \tag{3.1}$$

The natural measure of the information that we expect to gain from the label of an unlabeled example is the expected instantaneous information gain taken with respect to the probability that each one of the two labels occurs. Let p_0 be the probability that the label of x_m is 0, given that $c \in V_{m-1}$ and let V_m^0 be

the version space that results from the label x_m being 0. Define p_1 and V_m^1 in the corresponding way for the case $c(x_m) = 1$. We define the *expected information gain* of x_i, given V_{i-1}, to be:

$$\begin{aligned} \mathcal{G}(x_i|V_{i-1}) &\doteq -p_0 \log \frac{\Pr_{\mathcal{P}}(V_i^0)}{\Pr_{\mathcal{P}}(V_{i-1})} - p_1 \log \frac{\Pr_{\mathcal{P}}(V_i^1)}{\Pr_{\mathcal{P}}(V_{i-1})} \\ &= -p_0 \log p_0 - (1-p_0)\log(1-p_0) \doteq H(p_0) \ , \end{aligned} \tag{3.2}$$

where $H(p)$ denotes the Shannon information content of a binary random variable whose probability of being 1 is p. We shall use log base 2 in our definition and measure the expected information gain in *bits*.[1] The maximal information gain from a single label is one bit. The information gain is thus a very attractive measure of the gain that can be expected from asking **Label** for the label of an example. However, as we show in Section 3.3, this measure, by itself, is not sufficient for guaranteeing a large reduction in the expected prediction error of the algorithm.

The "Gibbs" prediction rule is to predict the label of a new example x by picking a hypothesis h at random from the version space and labeling x according to it. The random choice of h is made according to the prior distribution $\mathcal{P}$ restricted to the version space. It is a simple observation (see [Haussler *et al.*, 1991b]), that the expected error of this prediction error is at most twice larger than the expected error of the optimal prediction rule which is the Bayes rule. We shall assume that our learning algorithm has access to an oracle, denoted **Gibbs**, which can compute the Gibbs prediction for a given example $x \in X$ and version space $V \subset \mathbf{C}$. Each time $\mathbf{Gibbs}(V, x)$ is called, a hypothesis $h \in \mathbf{C}$ is chosen at random according to the distribution $\mathcal{P}$ restricted to V, and the label $h(x)$ is returned. Note that two calls to **Gibbs** with the same V and x can result in different predictions. The main result of the paper is that a simple algorithm for learning using queries, that uses the Gibbs prediction rule, can learn some important concept classes with accuracy that is exponentially small in the number of calls to **Label**.

3.3 Two simple learning problems

In this section we discuss two very simple learning problems. Our goal here is to give examples of the concepts defined in the previous section and to show that selecting examples to be queries solely according to their expected instantaneous information gain is not a good method in general.

Consider the following concept class. Let $X = [0, 1]$, and let the associated probability distribution $\mathcal{D}$ be the uniform distribution. Let the concept class $\mathbf{C}$, consist of all functions of the form

$$c_w(x) = \begin{cases} 1, & w \leq x \\ 0, & w > x \end{cases} \ , \tag{3.3}$$

where $w \in [0, 1]$. We define the prior distribution of concepts, $\mathcal{P}$ to be the one generated by choosing w uniformly from $[0, 1]$.

The version space defined by the examples $\{\langle x_1, c(x_1)\rangle, \ldots, \langle x_m, c(x_m)\rangle\}$ is (isomorphic to) the segment $V_i = [\max(x_i|c(x_i) = 0), \min(x_i|c(x_i) = 1)]$. Let us denote by ξ_i the ratio of between the probabilities of

[1] Here, and elsewhere in the paper, $\log(\cdot)$, denotes the logarithm over base two, while $\ln(\cdot)$ denotes the logarithm over base e.

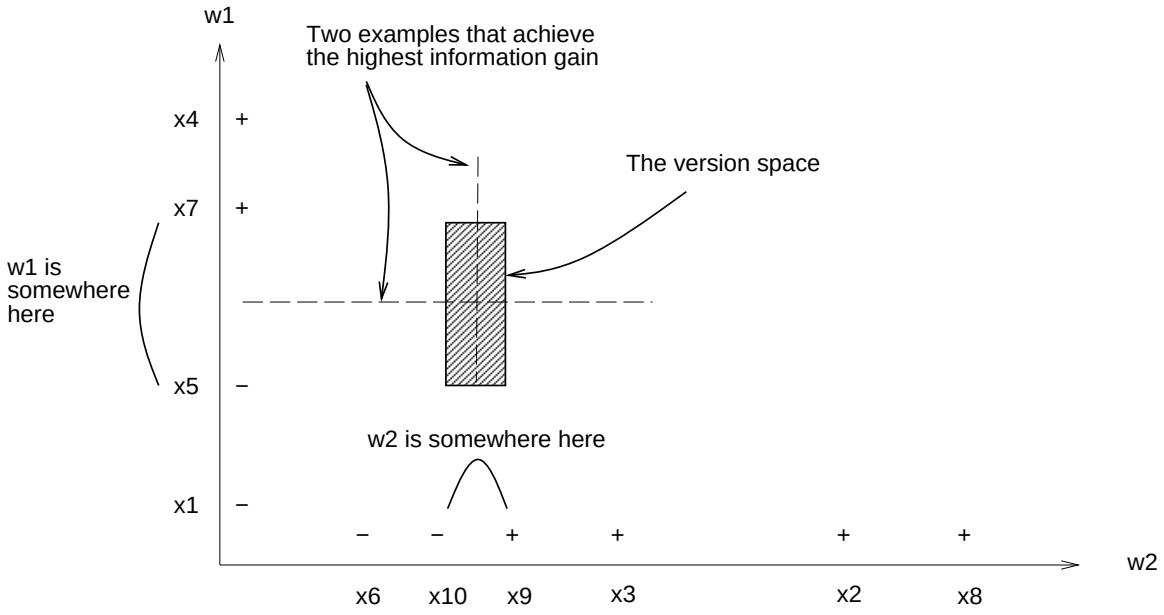


Figure 3.1: A figure of the version space and the examples that achieve maximal information gain for the two threshold learning problem defined below.

the version space before and after observing the ith example, i.e. $\xi_i = \Pr_{\mathcal{P}} V_i / \Pr_{\mathcal{P}} V_{i-1}$. The instantaneous information gain of the example $\langle x_i, c(x_i) \rangle$ is $\log \xi_i$. Given an *unlabeled* example, the expected instantaneous information gain from x_i is $H(\xi_i)$. Examples that fall outside the segment have a zero expected information gain, while the example that divides the segment into two equal parts obtains the highest possible expected information gain of one bit. This agrees with our intuition because the label of examples that fall outside the segment are already determined by previous labeled examples, while the label of the example that falls in the middle of the version space interval is least predictable. It is easy to calculate the probability of a prediction error for the Gibbs prediction rule for a given version space segment. This probability is equal to the length of the segment divided by three. Thus, if the learner asks for the label of the example located in the middle of the segment, it is guaranteed to half the error of the Gibbs prediction rule. In this case we see that asking the oracle **Label** to label the example that maximizes the expected information gain guarantees an exponentially fast decrease in the error of the Gibbs prediction rule. In contrast, the expected prediction error after asking for the labels of n randomly chosen examples is $O(1/n)$. The question is whether choosing queries according to their expected information gain is a good method in general, i.e. whether it always guarantees that the prediction error decreases exponentially fast to zero.

The answer to this question is negative, to see why this is the case consider the following, slightly more complex, learning problem. Let the sample space be the set of pairs in which the first element, i, is either 1 or 2, and the second element, z, is a real number in the range $[0, 1]$, i.e. $x \in X = \{1, 2\} \times [0, 1]$. Let $\mathcal{D}$ be the distribution defined by picking both i and z independently and uniformly at random. Let the concept class be the set of functions of the form

$$c_{\vec{w}}(i, z) = \begin{cases} 1, & w_i \leq z \\ 0, & w_i > z \end{cases}, \tag{3.4}$$

where $\vec{w} \in [0,1]^2$. The prior distribution over the concepts is the one generated by choosing $\vec{w}$ uniformly at random from $[0,1]^2$. In this case each example corresponds to either a horizontal or a vertical half plane, and the version space, at each stage of learning, is a rectangle (see Figure 3.3). There are always two examples that achieve maximal information gain, one horizontal and the other vertical. Labeling each one of those examples reduces the volume of the version space by a factor of two. However, the probability that the Gibbs rule makes an incorrect prediction is proportional to the length of the perimeter of the rectangular version space, and not to its volume. Thus, if the learner always chooses to ask queries of the same type, only one of the dimensions of the rectangle is reduced, and the perimeter length stays larger than a constant. Which implies that the prediction error also stays larger than a constant.

We conclude that the expected information gain of an unlabeled example is *not* a sufficient criterion for choosing good queries. The essential problem is that the distribution over the examples is completely ignored by this criterion. While one can easily find a specific solution for the given learning problem, we would like to have a general method that is sensitive to the distribution of the examples, and is guaranteed to work for a wide variety of problems. In the next section we present such a method.

3.4 The Query by Committee learning algorithm

Seung, Opper and Sompolinsky [Seung *et al.*, 1992] have devised an algorithm for learning by queries which they called "Query by Committee" and we shall refer to as the **QBC** algorithm. The algorithm uses as queries examples whose expected information gain is high, however, rather than *constructing* the examples, it *selects* the more informative examples from the random unlabeled examples that it gets from the oracle **Sample**.

The algorithm proceeds in iterations, in each iteration it calls **Sample** to get a random instance x. It then calls **Gibbs** twice, and compares the two predictions for the label of x. If the two predictions are equal, it rejects the instance and proceeds to the next iteration. If the two predictions differ, it calls **Label** with input x, and adds the labeled example to the set of labeled examples that define the version space. It then proceeds to the next iteration. In [Seung *et al.*, 1992] Seung et. al. treat the query by committee algorithm as an on-line learning algorithm, and analyze the rate at which the error of the two Gibbs learners reduces as a function of the number of queries made. In our work we prove general bounds both on the number of queries and on the number of random examples that the algorithm tests. In order to do that we consider a *batch* learning scenario, in which the learning algorithm is tested only after it has finished observing all of the training examples and has fixed its prediction hypothesis.

To do that we define a termination condition on the iterative process described above. When the algorithm reaches this a state that fulfills this condition it stops calling **Sample** and **Label** and uses the **Gibbs** oracle to *predict* the labels of the instances that it receives in the test phase. The termination condition is satisfies if a large number of consecutive instances supplied by **Sample** are all rejected.

We measure the quality of the predictions made by the algorithm in a way similar to that used in Valiant's PAC model. We define the expected expected error of the algorithm as the probability that its prediction of the label of a random instance disagrees with that of the true underlying concept. This probability is taken with respect to the random choice of the instance as well as the underlying concept.

We also allow the algorithm some small probability of failure to account for the fact that the sequence of instances that it observes during training is atypical.

We say that the learning algorithm is successful if its expected error, when trained on a typical sequence of instances, is small. More Precisely, we define two parameters, an accuracy parameter $1 > \epsilon > 0$ and a reliability parameter $1 > \delta > 0$. We use the term "training history" to describe a specific sequence of random instances and random coin flips used during learning a specific hidden concept. For each choice of the hidden concept, we allow a set of training histories that has probability δ to be marked as "atypical" training histories. Our requirement is that the expected error over the set of typical training histories is smaller than ϵ. The parameters ϵ and δ are provided to the learning algorithm as input and are used to define the termination criterion. Figure 3.2 gives a formal description of the algorithm. It is important to notice that the termination condition depends only on ϵ and δ, and not of any properties of the concept class. While the performance of the algorithm *does depend* on such properties, the algorithm can be used without prior knowledge of these properties.

It is easy to show that if **QBC** ever stops, then the error of the resulting hypothesis is small with high probability. That is because it is very unlikely that the algorithm stops if the probability of error is larger than ϵ (proof is given in Lemma 3.5.3). The harder question is whether **QBC** ever stops, and if it does, how many calls to **Sample** and to **Label** does it make before stopping? As we shall show in the following two sections, there is a large class of learning problems for which the algorithm will stop, with high probability, after $O(1/\epsilon \log 1/\delta\epsilon)$ calls to **Sample**, and $O(\log 1/\epsilon)$ calls to **Label**.

The committee filter tends to select examples that split the version space into two parts of comparable size, because if one of the parts contains most of the version space, then the probability that the two hypotheses will disagree is very small. Let us normalize the probability of the version space to one and assume that an example x partitions the version space into two parts with probabilities F and $1 - F$ respectively. Then the probability of accepting the example x as a query is $2F(1 - F)$ and the information gain from an example is $H(F)$. Both of these functions are maximized at $F = 0.5$ and decrease symmetrically to zero when F is increased to one or decreased to zero. It is thus clear that the queries of **QBC** have a higher expected information gain than random examples. However, it is not true in general that the expected information gain of the queries will always be larger than a constant,[2] moreover, as we have seen in the Section 3.3, queries with high information gain do not guarantee a fast decrease of the prediction error in general. Our proof of the performance of **QBC** consists of two parts. In the first part, given in Section 3.5, we show that a lower bound on the information gain of the queries *does* guarantee a fast decrease in the prediction error of **QBC**. In the second part, given in Section 3.6, we show that the expected information gain of the queries of **QBC** is guaranteed to be higher than a constant in some important cases.

[2]For example, consider the case in which the version space contains two disconnected sets in R^2, which are very far from each other, and assume that a random example is very likely to separate between these two sets. Suppose one of the sets has probability ϵ, while the other has probability $1 - \epsilon$. While most of the examples that separate between the two sets are rejected, the fraction that is accepted can still dominate all other examples. Thus the expected information gain is close to $H(\epsilon)$. As ϵ can be set arbitrarily small, the expected information gain can be arbitrarily close to zero. It seems that this type of version space can occur only very rarely but we do not know what are the necessary conditions.

Input: $\epsilon > 0$ - the maximal tolerable prediction error.

$\delta > 0$ - the desired reliability.

Gibbs- an oracle that computes Gibbs predictions.

Sample- an oracle that generates unlabeled examples.

Label- an oracle that generates the correct label of an example.

Initialize n - the counter of calls to **Label** – to 0, and set the initial version space, V_0, to be the complete concept class **C** .

Repeat until more than t_n consecutive examples are rejected. Where

$$t_n = \frac{1}{\epsilon} \ln \frac{\pi^2 (n+1)^2}{3\delta} \ ,$$

and n is the number of examples that have been used as queries so far.

1. Call **Sample** to get an unlabeled example $x \in X$ drawn at random according to $\mathcal{D}$.
2. Call **Gibbs**(V_n, x) twice, to get two predictions for the label of x.
3. **If** the two predictions are equal **then** reject the example and return to the beginning of the loop. (step 1)
4. **Else** call **Label**(x) to get $c(x)$, increase n by 1, and set V_n to be all concepts $c' \in V_{n-1}$ such that $c'(x) = c(x)$.

Output as the prediction hypothesis **Gibbs**(V_n, x).

Figure 3.2: Query by a committee of two

3.5 Relating information gain and prediction error for Query by Committee

In this section we prove that if the expected information gain from the queries used by **QBC** is high, then the prediction error of the algorithm is guaranteed to be exponentially small in the number of queries asked. We shall first define exactly what we mean by high information gain, and then give the theorem and its proof.

In our analysis we treat runs of the algorithm as initial segments of infinite runs that would have been generated if there was no termination criterion on the execution of the main loop in **QBC**. We denote by $\vec{X}$ the infinite sequence of unlabeled examples that would have been generated by calls to **Sample**. We use an infinite sequence of integer numbers $I = \{1 \leq i_1 < i_2 < \ldots\}$ to refer to the sequence of indices of those examples that are selected by **QBC** from $\vec{X}$ and used as queries to **Label**. This set of examples is denoted $\vec{X}_I$. We denote by M the sequence of integers from 1 to m, and use $\vec{X}_M$ to denote the first m examples in $\vec{X}$. We use I_n to denote the first n indices in I. Finally, $\vec{X}_{I_n}$ indicates the first n examples that are used as queries, and $\vec{X}_{I \cap M}$ indicates the queries that are chosen from the first m unlabeled examples.

We now present the probabilistic structure underlying the query process. A point in the sample space Ω is a triple $\langle c, \vec{X}, I \rangle$. The probability distribution over this space is defined as follows. The target concept c is chosen according to $\mathcal{P}$, and each component in the infinite sequence $\vec{X}$ is chosen independently according to $\mathcal{D}$. Fixing c and $\vec{X}$, we define the distribution of the first n elements of I according the probability the

algorithm **QBC** chooses to call the oracle **Label** on the iterations indexed by I_n. It is easy to see that the distributions defined for different values of n are consistent with each other, thus we can define the distribution on I as the limiting distribution for $n \to \infty$. We denote the distribution we have defined on the triplets $\langle c, \vec{X}, I \rangle$ by Δ and use $\Pr_\Delta$ and E_Δ to indicate the probability and the expectation taken with respect to this distribution.

We now define formally what we mean when we say that the queries of **QBC** are informative.

Definition 2: *We say that the expected information gain of queries made by* **QBC** *for the learning problem* $\mathbf{C}, \mathcal{P}, \mathcal{D}$ *is uniformly lower bounded by* $g > 0$ *if the following holds.*

For the distribution over $\langle c, \vec{X}, I \rangle$ *that is generated by* $\mathcal{P}, \mathcal{D}$ *and* **QBC** *and for every* $n \geq 0$*, the expected instantaneous information gain from the* $n+1$*st query, given any sequence of previous queries and their answers, is larger than* g*. In our notation we can write this as the requirement that the following conditional expectation is larger than* g *almost everywhere:*

$$Pr_\Delta \left(E \left(\mathcal{G}(x_{i_{n+1}} | V(\langle \vec{X}_{I_n}, c(\vec{X}_{I_n}) \rangle)) \right) \;\; | \;\; \vec{X}_{I_n}, c(X_{I_n}) \right) > g \Big) = 1$$

In somewhat more intuitive terms, a uniform lower bound on the information means that for any version space that can be reached by **QBC** with non-zero probability, the expected information gain from the next query of **QBC** is larger than g. In Section 3.6 We shall prove uniform lower bounds on the information gain of **QBC** for some important learning problems.

We now give the theorem that relates the bound on the information gain of **QBC** to its expected prediction error.

Theorem 3.5.1: *If a concept class* $\mathbf{C}$ *has VC-dimension* $0 < d < \infty$ *and the expected information gain of queries made by* **QBC** *is uniformly lower bounded by* $g > 0$ *bits, then the following happens with probability larger than* $1 - \delta$ *over the random choice of the target concept, the sequence of examples, and the choices made by* **QBC***:*

- *The number of calls to* **Sample** *that* **QBC** *makes is smaller than*

$$m_0 = \max \left(\frac{4d}{e\delta}, \frac{160(d+1)}{g\epsilon} \max \left(6, \ln \frac{80(d+1)}{\epsilon \delta^2 g} \right)^2 \right) \; . \tag{3.5}$$

- *The number of calls to* **Label** *that* **QBC** *makes is smaller than*

$$n_0 = \frac{10(d+1)}{g} \ln \frac{4m_0}{\delta} \; ,$$

  In other words, it is an exponentially small fraction of the number of calls to **Sample**.[3]

- *The probability that the* **Gibbs** *prediction algorithm that uses the final version space of* **QBC** *makes a mistake in it prediction is smaller than* ϵ*.*

Before we proceed to prove the theorem, let us give a brief intuitive sketch of the argument (See Figure 3.3). The idea is that if a concept class is learnable then, after observing many labeled examples,

[3]Note that the number of calls to **Sample** must is $\Omega(d/\epsilon)$ ([Blumer *et al.*, 1989]), even if *all* of the instances are used as queries to **Label**.

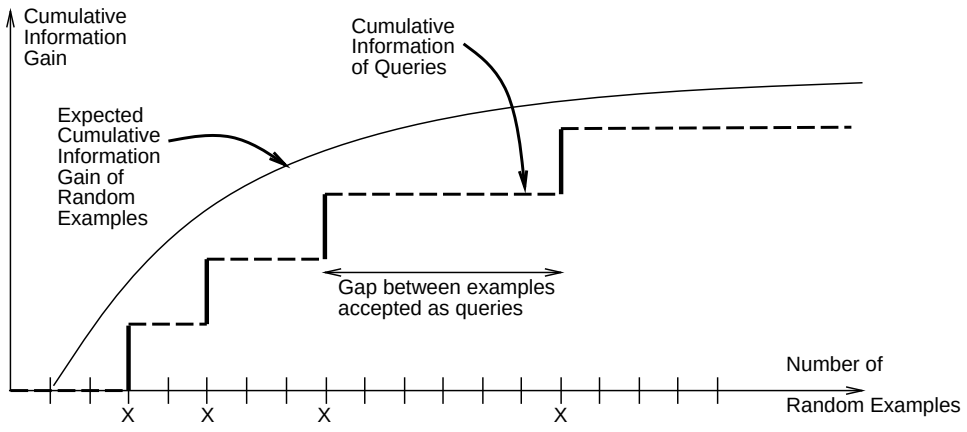


Figure 3.3: Each tag on the x axis denotes a random example in a specific typical sequence. The symbol X under a tag denotes the fact that the example was chosen as a query.

the conditional distribution of the labels of new examples is highly biased to one of the two labels. This means that the information gained from knowing the label of a random example is small. This, in turn, means that the increase in the cumulative information from a sequence of random examples becomes slower and slower as the sequence gets longer. On the other hand, if the information gained from the queries of **QBC** is lower bounded by a constant, then the cumulative information gain from the sequence of queries increases linearly with the number of queries. It is clear that the information from the labels of the queries alone is smaller than the information from the labels of all the examples returned by **Sample**. The only way in which both rates of increase can hold without violating this simple inequality is if the number of examples that are rejected between consecutive queries increases with the number of queries. As a result the termination criterion of **QBC** will hold, and the algorithm will output it's final prediction rule after a reasonably small number of queries. The prediction rule that is output is the Gibbs prediction rule, using the final version space that is defined by all the labeled examples seen so far. The probability of making a prediction error using this rule is, by definition, equal to the probability of a disagreement between a hypothesis that is randomly chosen according to the prior distribution restricted to the version space and a concept that is independently chosen according to the same distribution. This probability is also equal to the probability of a accepting a random example as a query when using this version space. The termination condition is fulfilled only if a large number of random examples are not accepted as queries, which implies that the probability of accepting a query or making a prediction mistake when using the final version space is small. We shall prove the theorem using the following three lemmas.

Lemma 3.5.2: *If the expected instantaneous information gain of the query algorithm is uniformly bounded by $g > 0$ bits, then*

$$Pr_{\Delta}(\mathcal{I}(\langle \vec{X}_{I_n}, c(\vec{X}_{I_n})\rangle) < \frac{g}{2}n) \leq e^{-\frac{g}{10}n} \tag{3.6}$$

Proof: The definition of a uniform lower bound on the expected information gain means that for any $n > 0$, for all sequence of of n queries $\langle \vec{X}_{I_n}, c(\vec{X}_{I_n})\rangle$, excluding possibly a set of measure zero, the expected information gain from the $n+1$st query is lower bounded by g. Put in another way, this means that the random variables

$$Y_i = \mathcal{I}(\langle \vec{X}_{I_i}, c(\vec{X}_{I_i})\rangle) - \mathcal{I}(\langle \vec{X}_{I_{i-1}}, c(\vec{X}_{I_{i-1}})\rangle) - g$$

form a sequence of sub-martingale differences. As the instantaneous information gain is bounded between 0 and 1, we get that $-g \leq Y_i \leq 1-g$. We can thus use Hoeffding's bound on the tails of bounded step sub-martingales [McDiarmid, 1989] [4] from which we know that for any $\epsilon > 0$

$$\Pr(\sum_{i=1}^{n} Y_i \leq -\epsilon n) \leq \left[(\frac{g}{g+\epsilon})^{g+\epsilon} (\frac{1-g}{1-g-\epsilon})^{1-g-\epsilon} \right]^n \ .$$

Setting $\epsilon = \lambda g$ and taking logs we get

$$\begin{gathered} \Pr(\textstyle\sum_{i=1}^{n} Y_i \leq -\lambda g n) \leq \\ \exp\left(\left(-(1+\lambda) g \ln(1+\lambda) + (1-(1+\lambda)g) \ln \frac{1-g}{1-(1+\lambda)g} \right) n \right) \leq \\ \exp\left((\lambda - (1+\lambda)\ln(1+\lambda)) g n \right) \ . \end{gathered}$$

Choosing $\lambda = 1/2$ we get the bound ∎

Lemma 3.5.3: *The probability that the predictions made by* **QBC** *are wrong (after its main loop has terminated) is smaller than ϵ with probability larger than $1 - \delta/2$.*

Proof: Assume that the probability of a wrong prediction is larger than ϵ. As discussed in the informal part of the proof, this implies that the probability of accepting a random example as a query with the final version space, is also larger than ϵ. It thus remains to show that the probability that **QBC** stops when the probability of accepting a query is larger than ϵ is smaller than $\delta/2$.

The termination condition of **QBC** is that all t_n examples tested after the nth query are rejected. If the probability of accepting a random example is larger than ϵ then this probability is smaller than $(1-\epsilon)^{t_n}$. From the definition of t_n we get that

$$(1-\epsilon)^{\frac{1}{\epsilon} \ln \frac{\pi^2 (n+1)^2}{3\delta}} \leq e^{-\ln \frac{\pi^2 (n+1)^2}{3\delta}} = \frac{3\delta}{\pi^2 (n+1)^2} \ .$$

Summing this probability over all possible values of n from zero to infinity we get the statement of the lemma. ∎

In [Haussler *et al.*, 1991b] it was shown that if the VC-dimension of a concept class is d, then the expected information gain from m random examples is bounded by $(d+1)\log(m/d)$. Here we show that the probability that the information gain is much larger than that is very small.

Lemma 3.5.4: *Assume a concept c is chosen at random from a concept class with VC dimension d. Fix a sequence of examples $\vec{X}$, recall that $\vec{X}_M$ denotes the first m examples. Then*

$$Pr_{c \in \mathcal{P}} \left(\mathcal{I}(\langle \vec{X}_M, c(\vec{X}_M)\rangle) \geq (d+1)(\log \frac{em}{d}) \right) \leq \frac{d}{em} \ . \tag{3.7}$$

[4] The bound as it appears in [McDiarmid, 1989] is given for martingales. However, it is easily checked that it is also true for super-martingales. Reversing the sign of the Y_i we get an equivalent theorem for sub-martingales.

Proof: From Sauer's Lemma [Sauer, 1972] we know that the number of different labelings created by m examples is at most $\sum_{i=0}^{d} \binom{m}{i} \leq (em/d)^d$. The expected cumulative information gain is equal to the entropy (base 2) of the distribution of the labels and is maximized when all the possible labelings have equal probability. This gives an upper bound of $d \log \frac{em}{d}$ on the expected cumulative information gain. Labelings that have cumulative information gain larger by a than this expected value, must have probability that is smaller by 2^a then the labels in the equipartition case. As the number of possible labelings remains the same, the total probability of all concepts that give rise to such labelings is at most 2^{-a}. Choosing $a = \log \frac{em}{d}$ we get the bound. ∎

Proof of Theorem 3.5.1 We consider a randomly chosen element of the event space $\langle c, \vec{X}, I \rangle$. Our analysis involves the first m_0 random examples presented to **QBC**, $\vec{X}_{M_0}$, and the first n_0 queries that **QBC** would choose if it never halts, $\vec{X}_{I_{n_0}}$. We denote the number of queries that **QBC** makes during the first m_0 examples by n, i.e. $n = |I \cap M_0|$. The claim of the theorem is that, with probability at least $1 - \delta$, the algorithm halts before testing the $m+1$st example, the number of queries it makes, n, is smaller than n_0, and the hypothesis it outputs upon halting has error smaller than ϵ. We shall enumerate a list of conditions that guarantee that all of these events occur for a particular random choice of examples and of internal randomization in **QBC**. By showing that the probability of each of those conditions to fail is small we get the statement of the theorem.

The conditions are:

1. The cumulative information content of the first n_0 queries is at least $gn_0/2$.
   From Lemma 3.5.2 we get that in order for this condition to hold with probability larger than $1 - \delta/4$ it is sufficient to require that
   $$n_0 \geq \frac{10}{g} \ln \frac{4}{\delta} \ . \tag{3.8}$$

2. The cumulative information content from the first m_0 examples is at most $(d+1)(\log \frac{em_0}{d})$.
   From Lemma 3.5.4 we get that in order for this condition to hold with probability larger than $1 - \delta/4$ it is sufficient to require that
   $$m_0 \geq \frac{4d}{e\delta} \ . \tag{3.9}$$

3. The number of queries made during the first m_0 examples, n, is smaller than n_0.
   The condition follows from conditions 1 and 2 if $\mathcal{I}(\langle \vec{X}_{I_{n_0}}, c(\vec{X}_{I_{n_0}}) \rangle) \geq \mathcal{I}(\langle \vec{X}_{M_0}, c(\vec{X}_{M_0}) \rangle)$. This is because if $n \geq n_0$ then the information gained from the queries asked during the first m_0 examples is larger than the total information gained from the m_0 examples, which is impossible. In order for $\mathcal{I}(\langle \vec{X}_{I_{n_0}}, c(\vec{X}_{I_{n_0}}) \rangle) \geq \mathcal{I}(\langle \vec{X}_{M_0}, c(\vec{X}_{M_0}) \rangle)$ to hold, it is sufficient to require that
   $$n_0 > \frac{2(d+1)}{g} (\log \frac{em_0}{d}) \ . \tag{3.10}$$

4. The number of consecutive rejected examples guarantees that the algorithm stops before testing the $m_0 + 1$st example.
   Notice that the threshold t_i increases with i. Thus if at least t_n consecutive examples from among

the first m_0 examples are rejected, the algorithm is guaranteed to halt before reaching the $m_0 + 1$st example. As there are $m_0 - n$ rejected examples, the length of the shortest run of ejected examples is at least $(m_0 - n)/(n + 1)$. We require that this expression is larger than t_n, and use the fact that condition 3 holds, i.e. that $n < n_0$. Using these facts it is sufficient to require that

$$m_0 \geq \frac{2(n_0 + 1)}{\epsilon} \ln \left[\frac{\pi^2}{3\delta} (n_0 + 1)^2 \right] \ . \tag{3.11}$$

5. The Gibbs prediction hypothesis that is output by the **QBC** has probability smaller than ϵ of making a mistaken prediction.
   From Lemma 3.5.3 we get that the probability of this to happen is smaller than $\delta/2$.

We see that if Equations (3.8), (3.9), (3.10), and (3.11) hold, then the probability that any of the four conditions fails is smaller than δ.It thus remains to be shown that our choices of n_0 and m_0 guarantee that these equations hold. Combining Equations (3.8) and (3.10), we get that it is sufficient to require that $m_0 \geq 2$, $d \geq 1$, and

$$n_0 + 1 = \frac{10(d + 1)}{g} \ln \frac{4m_0}{\delta} \tag{3.12}$$

Plugging this choice of n_0 into Equation (3.11), we get the following requirement on m_0:

$$m_0 \geq \frac{40(d + 1)}{\epsilon g} \ln \frac{4m_0}{\delta} \ln \left[\frac{20(d + 1)}{\delta g} \ln \frac{4m_0}{\delta} \right] \ . \tag{3.13}$$

It is simple algebra to check that the following choice of m_0 and satisfies Equations (3.9) and (3.13):

$$m_0 = \max \left(\frac{4d}{e\delta}, \frac{160(d + 1)}{g\epsilon} \max \left(6, \ln \frac{80(d + 1)}{\epsilon \delta^2 g} \right)^2 \right) \ , \tag{3.14}$$

Equations (3.12) and (3.14) guarantee that the conditions 1-5 hold with probability at least $1 - \delta$. ∎

3.6 Concept classes that are efficiently learnable using QBC

In this section we show that there exists a uniform lower bound for some interesting geometric concept classes. Our main analysis is for a learning problem in which concepts are intersections of half-spaces with a compact and convex subset of R^d. In this case the concept class itself can be represented as a compact and convex subset of R^d and each example partitions the concept class by a $d - 1$ dimensional hyperplane. We first prove that for the case in which both $\mathcal{D}$ and $\mathcal{P}$ are uniform there is a uniform lower bound on the information gain of **QBC** that does not depend on the dimension d. The proof is based on variational analysis of the geometry of the version space and is given in Section 3.6.1. In Section 3.6.2 we extend this proof to the non-uniform case. In Section 3.6.3 we use this result to prove a lower bound for perceptron learning under the uniform distribution.

3.6.1 Uniformly distributed half-spaces

In this subsection we prove a lower bound on the information gain for a simple geometric learning problem to which we shall refer as the "parallel planes" learning problem.

We define the domain, X, to be the set of all pairs of the form $(\vec{x}, t)$, where $\vec{x}$ is a vector in R^d whose length is 1, which we refer to as the "direction" of the example, and t is a real number in the range $[-1, +1]$, to which we refer as the offset. In other words $X = S^d \times [-1, +1]$, where S^d denotes the unit sphere around the origin of R^d. In this section we assume that the distribution $\mathcal{D}$ on X is uniform.[5] The concept class, $\mathbf{C}$, is defined to be a set of binary functions over X, parameterized by vectors $\vec{w} \in R^d, ||\vec{w}||_2 \leq 1$, that are defined as follows

$$c_{\vec{w}}(\vec{x}, t) = \begin{cases} 1, & \vec{w} \cdot \vec{x} \geq t \\ 0, & \vec{w} \cdot \vec{x} < t \end{cases} . \tag{3.15}$$

We assume that the prior distribution is uniform on B^d - the unit ball of radius one around the origin. This concept class is very similar to the class defined by the perceptron with variable threshold.[6] However, note that in this case the threshold, t, is part of the input, and not a parameter that defines the concept. This concept class is a bit strange, but as we shall see, the results we can prove for it can be used on the context of more natural concept classes such as the perceptron and thresholded smooth functions.

The reason that we can prove a uniform lower bound on the expected information gain of **QBC** for this concept class is that all the version spaces that can be generated when learning a concept from this concept class share some properties. The first property is each example, $(\vec{x}, t)$ cuts the version space by a plane that is orthogonal to the direction $\vec{x}$ and has offset t from the origin.[7] As t is uniformly distributed, the planes that cut the version space in each direction have a uniformly distributed offset that spans width of the version space in that direction. The second property is that all version spaces that can be generated when learning this concept class are bounded convex sets because they are defined as the intersection of a ball with a number of half-spaces.

As discussed in Section 3.4, both the expected information gain of an example and the probability that the example is accepted by **QBC** are quantities that depend on the ratio between the probabilities of the two parts of the version space that are created by the example. Based on these observations we can reduce our problem to a one dimensional problem. Fix a particular direction $\vec{x}$. Let $F_{\vec{x}} : [-1, +1] \rightarrow [0, 1]$ be the fraction of the version space, V, that is on one side of the plane defined by $\vec{x}$ and t, i.e.

$$F_{\vec{x}}(t) = \frac{\Pr_{c_{\vec{w}} \in \mathcal{P}} \left(c_{\vec{w}} \in V | c_{\vec{w}}(\vec{x}) \leq t \right)}{\Pr_{c_{\vec{w}} \in \mathcal{P}} \left(c_{\vec{w}} \in V \right)} . \tag{3.16}$$

We call F the *volume function* of the version space. The probability that **QBC** accepts the example $(\vec{x}, t)$ is $2F_{\vec{x}}(t)(1 - F_{\vec{x}}(t))$, and the expected information gain from the example is $H(F_{\vec{x}}(t))$. As t is uniformly distributed, the expected information gain from the examples whose direction is $\vec{x}$ is

[5] Actually, it is enough to assume that the distribution of the offset t is uniform for any direction $\vec{x}$. No assumption needs to be made regarding the distribution of $\vec{x}$.

[6] The perceptron concept class is defined as the following set of binary functions over the unit sphere

$$c_{\vec{w},t}(\vec{x}) \begin{cases} 1, & \vec{x} \cdot \vec{w} \geq t \\ 0, & \text{otherwise} \end{cases} .$$

[7] In the following discussion we ignore the distinction between the concepts in $\mathbf{C}$ and their parameterization, and refer to the concept $c_{\vec{w}}$ simply as the vector $\vec{w}$.

$$G(F_{\vec{x}}) = \frac{\int_{-1}^{+1} F_{\vec{x}}(t)(1 - F_{\vec{x}}(t))H(F_{\vec{x}}(t))\,dt}{\int_{-1}^{+1} F_{\vec{x}}(t)(1 - F_{\vec{x}}(t))\,dt} \ . \tag{3.17}$$

We now give the main technical theorem of this section. The theorem proves a lower bound on the value of $G(F_{\vec{x}})$. The proof is based on finding the convex version space that produces the smallest value of $G(F_{\vec{x}})$. This body is constructed of two equivalent cones connected at their bases. Barland [Barland, 1992, Theorem 5], analyzes a similar problem. He finds the convex body that achieves the minimal value of the functional $\int_{-1}^{+1} \min(F_{\vec{x}}(t), 1 - F_{\vec{x}}(t))\,dt$. Interestingly enough, he finds that the same body produces the minimal value of this functional.

Theorem 3.6.1: *The value of the functional $G(F_{\vec{x}})$ for the parallel planes learning problem, when the prior and input distributions are uniform, is at least $1/9 + 7/(18 \ln 2) > 0.672$ bits, independent of the dimension d.*

Proof: The proof is based on a variational analysis of the functional G. We shall show that there is one volume function that corresponds to a convex body which minimizes this functional. Using a number of transformations we shall show that the volume function of any other convex body can be transformed to a volume function of a different convex body which obtains a smaller value of G.

We shall bound the value of $G(F_{\vec{x}})$ independently of the direction $\vec{x}$. Our bound depends only on the fact that the version space is a bounded convex set in R^n and that the distribution in it is uniform. We thus drop the subscript $\vec{x}$ from $F_{\vec{x}}(\cdot)$. As $F(-1) = 0$, $F(+1) = 1$, and $H(1) = H(0) = 0$, we will, without loss of generality, extend the definition of $F(t)$ to all of R by defining it to be zero for $t \leq -1$ and one for $t \geq 1$. We then redefine the integrals in the definition of $G(F)$ in Equation (3.17) to be from from $-\infty$ to ∞. It is easy to check that $G(F(t)) = G(F(at + b))$ for any $a, b \neq 0$. Thus, without loss of generality, we will consider the set of volume functions to be the set of all monotone non-decreasing functions, F, such that $F(0) = 1/2$ and there exist $a_- \leq 0 \leq a_+$ such that $F(a_-) = 0$ and $F(a_+) = 1$.

We first show that for any volume function $F(t)$, one of the two functions:

$$F_-(t) = \begin{cases} F(t) & t \leq 0 \\ 1 - F(-t) & t > 0 \end{cases} \ ,$$

$$F_+(t) = \begin{cases} 1 - F(-t) & t \leq 0 \\ F(t) & t > 0 \end{cases} \ ,$$

gives a lower value of G. We can partition the integrals in the numerator and the denominator of Equation (3.17) into two parts:

$$G(F) = \frac{\int_{-\infty}^{+\infty} F(x)(1 - F(x))H(F(x))\,dx}{\int_{-\infty}^{+\infty} F(x)(1 - F(x))\,dx}$$

$$= \frac{\int_{-\infty}^{0} F(x)(1 - F(x))H(F(x))\,dx + \int_{0}^{+\infty} F(x)(1 - F(x))H(F(x))\,dx}{\int_{-\infty}^{0} F(x)(1 - F(x))\,dx + \int_{0}^{+\infty} F(x)(1 - F(x))\,dx} \ .$$

As all the integrals are positive it is easy to get a contradiction if $G(F)$ is smaller than both

$$G(F_+) = \frac{2\int_{0}^{+\infty} F(x)(1 - F(x))H(F(x))\,dx}{2\int_{0}^{+\infty} F(x)(1 - F(x))\,dx} \ . \tag{3.18}$$

and

$$G(F_-) = \frac{2\int_{-\infty}^{0} F(x)(1-F(x))H(F(x))\,dx}{2\int_{-\infty}^{0} F(x)(1-F(x))\,dx}\ , \tag{3.19}$$

Thus at least one of the expressions is smaller than $G(F)$. In the remainder of the proof we shall restrict our attention to volume functions F that are created in this way. For every convex body, whose volume function is F, we define the volume functions F_- and F_+. These volume functions do not necessarily correspond to convex functions. Rather, they correspond to bodies that are symmetric around the plane defined by $t = 0$, such that each of the two symmetric parts is a convex body. We shall call this type of volume functions "projection symmetric" and denote them by PS. Formally, F is a projection symmetric volume function if there exists a convex body whose volume function is $\hat{F}$, such that $F = \hat{F}_+$ or $F = \hat{F}_-$. As in this case $F(t) = 1 - F(-t)$, expressions (3.19) and (3.18) are both equal to $G(F)$ and we redefine $G(F)$ as expression (3.18). We shall show that the projection symmetric volume function that attains the lowest value of G corresponds to a convex body and in this way show that this is the volume function corresponding to a convex body that minimizes G.

In order to find the PS volume function that minimizes $G(F)$ we use infinitesimal variational transformations of F that decrease the value of G. We find it convenient to define simple transformations of F and H. Define $K(t) = F(t)(1 - F(t))$, and $Q(x) = H(1/2 - \sqrt{1-4x}/2)$, then Equation (3.17) can be written in a slightly simpler form as

$$G(K) = \frac{\int_0^{+\infty} K(t)Q(K(t))\,dt}{\int_0^{+\infty} K(t)\,dt}\ . \tag{3.20}$$

The changes in $G(K)$ for small changes in the function K can be approximated by a linear functional, called the Fréchet derivative,[8]

as follows

$$G(K+\Psi) = G(K) + \int_0^{+\infty} \nabla G(t)\Psi(t)\,dt + o\left(\int_0^{+\infty} \Psi(t)^2\,dt\right)\ ,$$

where

$$\begin{aligned}\nabla G(t) &= \frac{\int_0^{+\infty} K(s)\,ds\ \frac{\partial}{\partial K(t)}\left(K(t)Q(K(t))\right) - \int_0^{+\infty} K(s)Q(K(s))\,ds\ \frac{\partial}{\partial K(t)}K(t)}{\left(\int_0^{+\infty} K(s)\,ds\right)^2} \\ &= \frac{1}{\int_0^{+\infty} K(s)\,ds}\left[Q(K(t)) + K(t)\frac{\partial}{\partial K(t)}Q(K(t)) - G(K)\right]\end{aligned} \tag{3.21}$$

We first consider the behavior of the sum of the first two terms in the square brackets. Denote $K(t)$ by y. A direct calculation shows that $Q(y) + y\frac{\partial}{\partial y}Q(y)$ is a strictly increasing function of y in the range $0 \le y \le 1/4$, which is the range of $K(t)$. It is 0 for $y = 0$ and 1 for $y = 1/4$.

As $0 \le G(K) \le 1$ the third term is in the range of the sum of the first two terms. As $K(t)$ is decreasing for positive t, it implies that there is some point $w > 0$, which is a function of K, such that for all $0 \le t \le w$, $\frac{\partial}{\partial K(t)}G(K(t)) > 0$, and for all $t > w$, $\frac{\partial}{\partial K(t)}G(K(t)) < 0$. The parameter w is of critical importance in the rest of the paper, and we shall refer to it is the "pivot point". In terms of the volume function F, for $t > 0$,

[8] Details on how the Fréchet derivative is defined and calculated can be found in standard books on variational analysis, such as [Smith, 1985].

F increases when K decreases and vice versa. Thus if the variational function $\Psi(t)$ is non-negative for points below the pivot point, non-positive for points above the pivot point, and $\int_0^{+\infty} \Psi(t)^2\, dt$ is sufficiently small then $G(K(t)+\Psi(t)) < 0$ as desired.

It remains to show how the variation functions $\Psi(t)$ can be chosen in a way that preserves the PS properties of the volume function. We define $f(t) = \frac{dF(t)}{dt}$, i.e. $f(t)$ is the $d-1$st dimensional volume of the slice of the version space located at t. We define the function $r(t)$ to be $\sqrt[d-1]{f(t)}$. One of the d dimensional bodies whose volume function gives rise to the function r is the body of revolution whose surface is created by revolving the function $r(t)$ around the t axis. For this reason $r(t)$ is also called the *radius function* of the body. The following lemma gives an important property of the radius function of convex bodies.

Lemma 3.6.2: *The radius function that corresponds to convex bodies is concave on its support set.*

Proof: Let us denote by S_t the convex body in R^{d-1} that is defined by the slice of the version space located at t. Clearly, $f(t)$ is the volume of S_t.

We define the linear combination of two bodies, A and B as:

$$\lambda_1 A + \lambda_2 B = \{\lambda_1 a + \lambda_2 b | a \in A,\ b \in B\}\ ,$$

where $\lambda_1, \lambda_2 \in R$. An immediate result of the convexity of the version space is that for any $t_1, t_2 \in R$, and any $0 \leq \lambda_1, \lambda_2 \leq 1$ such that $\lambda_1 + \lambda_2 = 1$

$$\lambda_1 S_{t_1} + \lambda_2 S_{t_2} \subseteq S_{\lambda_1 t_1 + \lambda_2 t_2}\ .$$

Using the terminology of the theory of convex bodies, we can say that the set of bodies S_t, parameterized by $t \in R$ is a (one-parameter) concave family of bodies.[9]

The Brunn Minkowski theorem states that, for bodies in R^n, *"the n-th root of the volume of the bodies of a linear or concave family is a concave function of the family of parameters"* ([Bonnesen and Fenchel, 1987],Subsection 48). In our case, $n = d-1$ and the family is a concave family of a single parameter. We thus get the statement of the lemma as a special case of the Brunn Minkowski theorem. ∎

We shall use $G_d(r)$ to denote $G(F)$ where F is the volume function of a d dimensional body whose radius function is r. We shall show that the radius function that corresponds to a PS volume function and minimizes $G(r)$ is (up to a normalization factor c_d that does not change $G_d(r)$):

$$r^*(t) = c_d \max(0, 1-|t|) \tag{3.22}$$

The revolution body that corresponds to this radius function is constructed of two equivalent cones joined at their bases. As this is a convex body we conclude that it is the convex body that minimizes $G(F)$. One can compute $G_d(r^*)$ for any fixed d by solving the integral in Equation (3.17) as follows. In this case we find it more convenient to use the integral over the negative half of the line as defined in Equation (3.18). The volume function in the range $-1 \leq t \leq 0$ is $F_d^*(t) = \int_{-\infty}^{t} (r^*(s))^{d-1} ds = (1+t)^d/2$ and it is 0 for $t < 0$. Plugging this into Equation (3.18) we get

$$G(F_d^*) = \frac{\int_{-1}^{0} \frac{(1+t)^{d+1}}{2}(1 - \frac{(1+t)^{d+1}}{2}) H(\frac{(1+t)^{d+1}}{2})\, dt}{\int_{-1}^{0} \frac{(1+t)^{d+1}}{2}(1 - \frac{(1+t)^{d+1}}{2})\, dt} = \frac{\int_0^{1/2} F^{1/d}(1-F)\mathcal{H}(F) dF}{\int_0^{1/2} F^{1/d}(1-F) dF}\ , \tag{3.23}$$

[9] For the definition of a convex family of bodies see ([Bonnesen and Fenchel, 1987],Subsection 24).

which can be shown by direct calculation to decrease as $d \to \infty$. Which gives the general lower bound of

$$G(F_d^*) > \frac{\int_0^{1/2}(1-F)\mathcal{H}(F)dF}{\int_0^{1/2}(1-F)dF} = \frac{1}{9} + \frac{7}{18\log 2} \ . \tag{3.24}$$

And this gives the statement of the theorem.

What remains to be shown is the existence of variations that decrease $G(F)$ for all volume functions but F^*, that preserve the PS properties of F. As the characterization for volume functions of convex bodies is given in terms of the radius function we describe the variations in terms of adding a variation function, $\psi(t)$ to the radius function $r(t)$. As we are restricting ourselves to volume functions, it is enough to define $\psi(t)$ for $0 \le t < \infty$.

Let us enumerate the requirements on the radius variation function $\psi(t)$, and on the corresponding volume variation function $F(t) + \Psi(t) = \int_0^t (r(s) + \psi(s))^{d-1}\, ds$.

1. We need $F(|t|) + \Psi(|t|)$ to be a PS volume function. For this to hold we require that $r(t) + \psi(t)$ is a positive concave function that is nonzero only on a bounded segment $[0, c]$, $c < \infty$.
2. We need to guarantee that $\int_0^\infty \nabla G(t)\Psi(t)\, dt < 0$. For that to hold we require that $\Psi(t)$ is non-positive for all $0 \le t \le w$ and non-negative for all $t > w$. Where w is the pivot point for the volume function F.
3. For any given $\epsilon > 0$ we should be able to find a radius variation function $\psi(t)$ such that the change in the corresponding volume function is as small as is desired $\epsilon > \int_0^{+\infty} \Psi(t)^2\, dt > 0$.

We describe three families of variational functions. For any radius function r that corresponds to a volume function in PS and is not equal to $r^* = c_d \max(0, 1 - |t|)$, one of these variations applies, showing that there exists $r'(t)$ such that $G_d(r') < G_d(r)$. The variations are constructed geometrically. Below is a list of the constructions that should be read alongside Figure 3.4. The basic idea in all three transformations is to "move" volume from place to place along the projection direction, in such a way that for each point t in a particular range, volume is moved only from one the right of the points to their left or vice versa. It is easy to check that each of the conditions 1-3 holds for each of those transformations. In the descriptions below we shall refer to volume changes are caused by increasing or decreasing the radius function, note that these are changes in the d-dimensional volume of the body whose volume function corresponds to the radius function, and not in the two dimensional area described by the changes in the graph. The transformations thus depend on the dimension of the actual body, however, the qualitative form of the transformation remains the same for all dimensions. Each transformation takes a parameter λ, which is a positive number that is set small enough so that condition 3 holds.

1. If r is not linear in the range $0 \le t \le w$ then transformation 1 is used (see Figure 3.4(a)):
   (a) Let A be the point $(w, r(w))$, select a point A' on the curve defined by r to the left of A so that the volume decrease caused by changing the curve[10] $A \frown A'$ to the cord $A - A'$ is equal $\lambda/2$.

[10] We use $A - B$ to denote the line segment between the points A and B, and $A \frown B$ to denote the segment of a curve that connects A and B. We also use the shorthand $A - B \frown C - D$ to denote a the concatenation of a line segment, a curve segment, and another line segment.

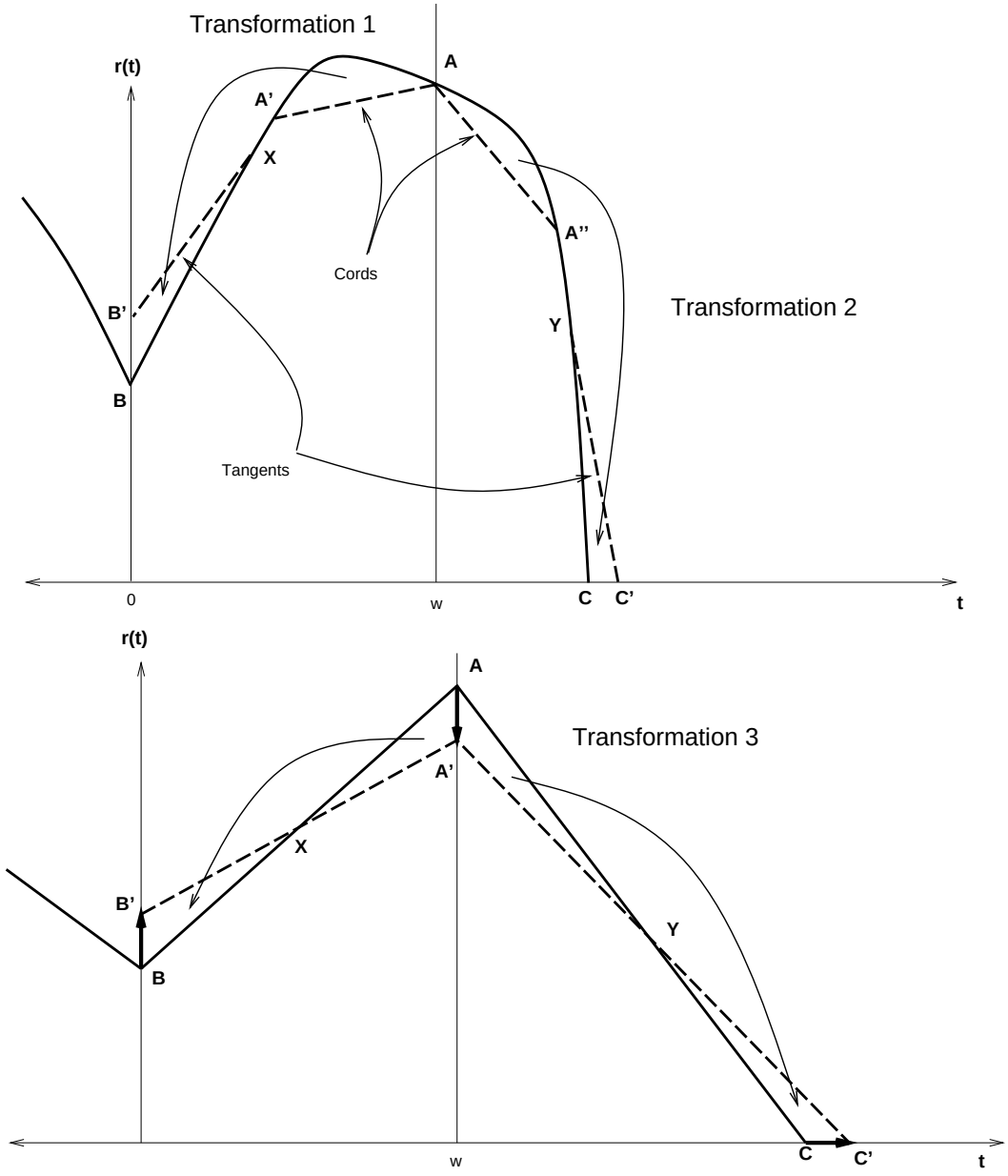


Figure 3.4: The variational transformations

(b) Let B be the point $(0, r(0))$, select a point B' slightly above B and connect it to the (unique) point X on the curve so that the curve $B - X \frown A' - A$ is concave. Choose B' so that the volume increase caused by changing the curve $B \frown X$ to the line $B - X$ is $\lambda/2$.

Set λ_0 small enough so that this construction is possible for all $0 < \lambda < \lambda_0$.

Note that for each point $0 \leq t \leq w$, at least one of the two following conditions hold: either volume is only removed from the right of t, or volume is only added to the left of t. This implies that the volume function, $F(t)$, increases in this range. Because the amount of volumes that are removed and added are equal, $F(t)$ does not change for t outside the range $[0, w]$. This implies that condition 2

holds.

2. If r does not decrease linearly to zero for $t \geq w$ then transformation 2 is used (see Figure 3.4(a)):
   (a) Select A'' on the curve to the right of A so the volume decrease that is caused by changing $A \frown A''$ to $A - A''$ is $\lambda/2$.
   (b) Let C be the point at which the curve meets the horizontal axis. Select C' slightly to the right of C and connect it to the point Y on the curve so that the curve $C' - Y \frown A'' - A$ is concave. Choose C' so that the volume increase caused by changing $C' - C - Y$ to $C' - Y$ is $\lambda/2$.

   Set λ_0 small enough so that this construction is possible for all $0 < \lambda < \lambda_0$.

   An argument similar to the one used in transformation 1 holds in this case for $t > w$.
3. If neither condition 1 nor 2 holds, and the slopes of the two linear segments are not equal (i.e. $r \neq r^*$), then transformation 3 is used (see Figure 3.4(b)):
   (a) A point A' slightly below A is chosen.
   (b) A point B' slightly above B is chosen so that there is no net change in the volume when changing $A - B$ to $A' - B'$.
   (c) A point C' slightly to the right of C is chosen so that there is no net change in the volume when changing $A - C$ to $A' - C'$.
   (d) The movement from A to A' is chosen do that the change in the volume caused by each of the four changes in r: $B - X$ to $B' - X$, $A - X$ to $A - X'$, $A - Y$ to $A' - Y$ and $C - Y$ to $C' - Y$ is equal to $\lambda/4$

   In this case the volume function is changed on both sides of the pivot point. Arguments similar to the one used in transformation 1 shows that condition 2 is met.

The only PS radius functions to which none of those transformations apply is r^*, thus finishing the proof of the theorem.

∎

3.6.2 Relaxing the uniformity constraints

In this section we show that the results of Theorem 3.6.1 can be extended to cases where the prior and input distributions are not exactly uniform. We use the following definition

Definition 3: *We say that a density $\mathcal{D}'$ is* within λ *of $\mathcal{D}$ if for every measurable set A, we have that $\lambda \leq Pr_{\mathcal{D}}(A)/Pr_{\mathcal{D}'}(A) \leq 1/\lambda$.*

Using this definition, we get the following extension of Theorem 3.6.1:

Theorem 3.6.3: *The value of the functional $G(F)$ for the parallel planes learning problem, when the prior distribution is within $\lambda_{\mathcal{P}}$ of uniform and the input distribution is within $\lambda_{\mathcal{D}}$ of uniform, is at least $\lambda_{\mathcal{P}}^4 \lambda_{\mathcal{D}}(1/9 + 7/(18 \ln 2)) > 0.672 \lambda_{\mathcal{P}}^4 \lambda_{\mathcal{D}}$ bits, independent of the dimension d.*

Proof:

We first prove the dependence on the uniformity of the input distribution, as measured by $\lambda_{\mathcal{D}}$. In general, any distribution $\mathcal{D}$ that is within $\lambda_{\mathcal{D}}$ of the uniform distribution μ can be written as weighted sum of the form $\lambda_{\mathcal{D}}\mu + (1-\lambda_{\mathcal{D}})\nu$ where ν is some other distribution. Fix the version space and *any* prior distribution, let the distribution of examples be $\mathcal{D} = \lambda_{\mathcal{D}}\mu + (1-\lambda_{\mathcal{D}})\nu$ and let g_μ, g_ν be the expected information gains when the examples are generated according to μ or ν respectively. As $g_\nu > 0$ we get that the expected information gain when $\mathcal{D}$ is within $\lambda_{\mathcal{D}}$ of uniform is at least $\lambda_{\mathcal{D}}$ times the expected information gain when $\mathcal{D}$ is uniform.

The analysis of the dependence on $\lambda_{\mathcal{P}}$ is more involved. We go back to the analysis of an arbitrary projection of a convex body from the proof of Theorem 3.6.1. The main idea there was to show transformations that increase or decrease the volume function in particular ranges, in a way that decreased the expected information gain. There, the transformation involved changing the shape of the body. Here we present a transformation that changes the density of the prior distribution inside the version space.

We fix a convex body and a direction $\vec{x}$ along which this body is projected. We denote by $\rho(t)$ the average density along the slice of body which is defined by the example $(\vec{x}, t)$. The relation between the volume function F, and the radius function r is now

$$F_d(t) = \int_{-\infty}^{t} (r(s))^{d-1}\rho(s)ds \ .$$

We search for a density distribution of the points in the body, which is within $\lambda_{\mathcal{P}}$ of the uniform distribution, and minimizes the expected information gain from (uniformly distributed) examples whose direction is $\vec{x}$. Note that the symmetrization argument used in the proof of Theorem 3.6.1 holds for this case too, and we can thus restrict ourselves to functions r and ρ that are defined only over the positive reals. From the variational derivative of $F(t)$ for $t \geq 0$ that we computed in Equation (3.21), we know that $G(F)$ decreases if $F(t)$ is increased for some $t \leq w$ or if $F(t)$ is decreased for some $0 \leq t \leq w$. As we allow deviations from the uniform prior distribution we can change F without changing the form of the convex body. We shall now give a variation of ρ that changes $\rho(t)$ in the range $0 \leq t \leq w$ in a way that decreases $G(F)$. As this variation can be applied to any ρ that does not have a specific step-like form in this range, we get that this step-like form of ρ achieves the minimal value of $G(F)$ for this fixed body and $\mathcal{P}$ that is within $\lambda_{\mathcal{P}}$ of uniform. A similar argument can be used to show that $\rho(t)$ must also have a stepwise form in the range $w \leq t$.

Assume that there exist $0 < t_1 < t_2 < w$ and $\epsilon, \delta > 0$ such that $0 \leq t_1 - \epsilon < t_1 + \epsilon \leq t_2 - \epsilon < t_2 + \epsilon < w$, and such that for all $t \in [t_1 - \epsilon, t_1 + \epsilon]$, $\rho(t) < 1/\lambda_{\mathcal{P}} - \delta$, and for all $t \in [t_2 - \epsilon, t_2 + \epsilon]$, $\rho(t) > \lambda_{\mathcal{P}} + \delta$. We add to $\rho(t)$ the following variation function:

$$\psi(t) = \begin{cases} +\delta_1, & t_1 - \epsilon \leq t \leq t_1 + \epsilon \ , \\ -\delta_2, & t_2 - \epsilon \leq t \leq t_2 + \epsilon \ , \ , \\ 0, & \text{otherwise} \end{cases}$$

where δ_1, δ_2 are chosen so that $\delta \geq \delta_1, \delta_2 > 0$ and

$$\frac{\delta_1}{\delta_2} = \frac{\int_{t_1-\epsilon}^{t_1+\epsilon} (r(s))^{d-1}ds}{\int_{t_2-\epsilon}^{t_2+\epsilon} (r(s))^{d-1}ds} \ .$$

This insures that the volume function does not change outside the range $[t_1 - \epsilon, t_2 + \epsilon]$.

It is easy to check that $\rho(t) + \psi(t)$ corresponds to a density distribution that is within $\lambda_{\mathcal{P}}$ of the uniform distribution. Changing the density distribution from $\rho(t)$ to $\rho(t) + \psi(t)$ decreases $F(t)$ in the range $[t_1 - \epsilon, t_2 + \epsilon]$ and does not change $F(t)$ anywhere else. Thus this change decreases $G(F)$. It is also easy to check that this variation cannot be applied to ρ if and only if there exists $0 \leq a \leq w$ such that $\rho(t) = 1/\lambda_{\mathcal{P}}$ for $0 \leq t < a$ and $\rho(t) = \lambda_{\mathcal{P}}$ for $a < t \leq w$. From this argument and a similar argument for the range $t \geq w$ we get that the density function that minimizes $G(F)$ must be of the form

$$\rho^*(t) = \begin{cases} 1/\lambda_{\mathcal{P}}, & 0 \leq t \leq a \text{ or } b \leq t \\ \lambda_{\mathcal{P}}, & a \leq t \leq b \end{cases} , \tag{3.25}$$

where $0 \leq a \leq w \leq b$. We do not have a simple variational argument for determining the exact value of a and b, however, as we shall see, we can lower bound the information gain without this explicit knowledge.

We have thus found the form of the density function that minimizes the information gain for a specific body (and a specific projections). Suppose now that we fix the function ρ and vary the shape of the body, i.e. the radius function r. Going through the construction of the variational functions ψ in the proof of Theorem 3.6.1, we see that the same construction steps hold verbatim, although special attention needs to meaning of the expression "the volume decrease is equal to x" as the volume is now defined in terms of the non uniform distribution specified by ρ.

The combination of these two arguments shows that the smallest value of $G(F)$ is attained for the radius function r^* specified in Equation (3.22), and the average density function ρ^*. It remains to compute a lower bound on $G(F)$ based on these two facts. This is done by bounding the ratio between the values of $G(F)$ for the uniform prior and the non uniform prior cases.

We change the integration variable in Equation (3.19) from x to $F(x)$:[11]

$$G(F) = \frac{\int_0^{1/2} F(1-F)\, H(F)\, \frac{dx}{dF} dF}{\int_0^{1/2} F(1-F)\, \frac{dx}{dF} dF} . \tag{3.26}$$

When written in this form, the dependence of $G(F)$ on the r and ρ enters the equation through the derivative dx/dF. By bounding the ratio between the values that this derivative attains in the uniform and the non-uniform cases, we can bound the ratio between the values that $G(F)$ attains for the uniform and the non-uniform prior distributions.

The volume function that corresponds to the uniform prior distribution is, for $-1 \leq x \leq 0$, $F_{\text{unif}}(x) = (1+x)^d/2$. The volume function that corresponds to the prior distribution defined by ρ^* is

$$F_{\text{non-unif}}(x) = \frac{1}{2} \begin{cases} \lambda_{\mathcal{P}}^{-1}(1+x)^d, & -1 \leq x \leq -b, \\ \lambda_{\mathcal{P}}(1+x)^d + c, & -b \leq x \leq -a, \\ \lambda_{\mathcal{P}}^{-1}(1+x)^d + 1 - \lambda_{\mathcal{P}}^{-1}, & -a \leq x \leq 0 \end{cases} \tag{3.27}$$

Where $c \geq 0$ is defined by matching the two definitions of $F(-b)$.

[11] Recall that we are considering the symmetric case, so $F_+ = F_- = F$.

Taking the derivatives of F_{unif} and $F_{\text{non-unif}}$ we get the following equation for their ratio:

$$\frac{\left(\frac{dx}{dF}\right)_{\text{non-unif}}}{\left(\frac{dx}{dF}\right)_{\text{unif}}} = \begin{cases} \lambda_{\mathcal{P}}^{1/d}, & 0 \le F \le F(-b), \\ \lambda_{\mathcal{P}}^{-1/d}\left(\frac{2F}{2F-c}\right)^{1-1/d}, & F(-b) \le F \le F(-a), \\ \lambda_{\mathcal{P}}^{1/d}\left(\frac{2F}{2F+\lambda_{\mathcal{P}}^{-1}-1}\right)^{1-1/d}, & F(-a) \le F \le 1/2 \end{cases} \tag{3.28}$$

Using the facts that $\lambda_{\mathcal{P}} \le 1$, $c \ge 0$, and $d \ge 2$ we can bound the ratio of the derivatives for each of the three cases. For the range $-1 \le x \le -b$ we get that

$$\lambda_{\mathcal{P}} \le \lambda_{\mathcal{P}}^{1/d} \le \frac{\left(\frac{dx}{dF}\right)_{\text{non-unif}}}{\left(\frac{dx}{dF}\right)_{\text{unif}}} \le 1 \ . \tag{3.29}$$

For the range $-b \le x \le -a$ we get, using the fact that F is monotone non-decreasing, that

$$1 \le \frac{2F(-a)}{2F(-a)-c} \le \frac{2F(x)}{2F(x)-c} \le \frac{2F(-b)}{2F(-b)-c} = \frac{\lambda_{\mathcal{P}}^{-1}(1-b)^d}{\lambda_{\mathcal{P}}(1-b)^d} \le \lambda_{\mathcal{P}}^{-2} \ ,$$

which implies that in the range $-b \le x \le -a$,

$$1 \le \lambda^{-1/d} \le \frac{\left(\frac{dx}{dF}\right)_{\text{non-unif}}}{\left(\frac{dx}{dF}\right)_{\text{unif}}} \le \lambda_{\mathcal{P}}^{-2+1/d} \le \lambda_{\mathcal{P}}^{-2} \ . \tag{3.30}$$

Finally, for the range $-a \le x \le 0$, we get that

$$\lambda_{\mathcal{P}}^2 \le \frac{\lambda_{\mathcal{P}}(1-a)^d + c}{\lambda_{\mathcal{P}}^{-1}(1-a)^d} = \frac{2F(-a)}{2F(-a)+\lambda_{\mathcal{P}}^{-1}-1} \le 1$$

which implies that

$$\lambda_{\mathcal{P}}^2 \le \lambda_{\mathcal{P}}^{2-1/d} \le \lambda^{-1/d} \le \frac{\left(\frac{dx}{dF}\right)_{\text{non-unif}}}{\left(\frac{dx}{dF}\right)_{\text{unif}}} \le 1 \tag{3.31}$$

Combining the bounds from Equations (3.29), (3.30), and (3.31), and plugging them into Equation (3.28), we get that

$$\lambda_{\mathcal{P}}^2 \le \frac{\left(\frac{dx}{dF}\right)_{\text{non-unif}}}{\left(\frac{dx}{dF}\right)_{\text{unif}}} \le \lambda_{\mathcal{P}}^{-2}$$

Using this bound and Equation (3.26) we get that $G(F_{\text{non-unif}}) \ge \lambda_{\mathcal{P}}^4 G(F_{\text{unif}})$. This completes the proof of the theorem. ∎

3.6.3 Perceptrons

Using the tools we developed in the previous sections, we can prove that **QBC** is an efficient query algorithm for the perceptron concept class when the prior distribution and the distribution of examples are both close to uniform.

The perceptron concept class is defined as the following set of binary functions over the unit ball

$$c_{\vec{w}}(\vec{x}) \begin{cases} 1, & \vec{x} \cdot \vec{w} \geq 0 \\ 0, & \text{otherwise} \end{cases} , \tag{3.32}$$

where $\vec{w}, \vec{x} \in R^d$, $||\vec{w}||_2 = 1$ and $||\vec{x}||_2 \leq 1$. The prior distributions are within some constants from the uniform distributions over the respective sets. As each $\vec{w}$ is a point on the surface of a d dimensional sphere, the initial version space is isomorphic to the unit sphere.

In this section we prove that there exists a lower bound on the information gain of the queries of **QBC**. However, our proof technique requires that the initial version space is not the complete unit sphere, but is restricted to be within a cone. In other words, there has to exist a unit vector $\vec{w}_0$ such that for any $\vec{w} \in V_0$ the dot product $\vec{w} \cdot \vec{w}_0$ is larger than some constant $\alpha > 0$.

This condition is annoying. However, it is not hard to guarantee that this condition holds by using an initial learning phase, prior to the use of **QBC**, that does not use filtering but rather queries on all the random instances supplied by **Sample**. Using the results of Blumer et. al. we can bound the number of training examples that are needed to guarantee that the prediction error of an arbitrary consistent hypothesis is small (with high probability). As the distribution of the instances is close to uniform, a small prediction error implies that the hypothesis vector is within a small angle of the vector that corresponds to the target concept. The details of this argument are given in the following lemma.

Lemma 3.6.4: *Assume that the distribution of the instances $\mathcal{D}$ is within $\lambda_{\mathcal{D}}$ from the uniform distribution in the unit ball. Suppose m random instances are chosen according to $\mathcal{D}$, labeled according to $f_{\vec{w}_0}(\cdot)$ and used to find a hypothesis $f_{\vec{w}}(\cdot)$ that is consistent with all the labeled instances.*

If

$$m \geq \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{13}{\epsilon}\right) \quad \textit{where } \epsilon = \lambda_{\mathcal{D}}\cos^{-1}(\alpha)$$

then, with probability $1-\delta$ over the choice of the m random instances, $\vec{w} \cdot \vec{w}_0 \geq \alpha$.

Proof: If $\vec{w} \cdot \vec{w}_0 < \alpha$ then the angle between $\vec{w}$ and $\vec{w}_0$ is larger than $\cos^{-1}(\alpha)$. The examples on which $f_{\vec{w}}(\vec{x})$ is incorrect are those vectors in the unit ball for which $\vec{x} \cdot \vec{w} \geq 0$ and $\vec{x} \cdot \vec{w}_0 < 0$, or $\vec{x} \cdot \vec{w} < 0$ and $\vec{x} \cdot \vec{w}_0 \geq 0$. This defines a subset of the unit ball, constructed of two wedges, whose volume is at least $\cos^{-1}(\alpha)$ of the volume of the ball. As the distribution of the instances is within $\lambda_{\mathcal{D}}$ from the uniform distribution, the probability of this set is at least $\lambda_{\mathcal{D}}\cos^{-1}(\alpha)$.

On the other as the VC dimension of the d dimensional perceptron is d we can use the classical uniform convergence bounds from [Blumer *et al.*, 1989]. Theorem 2.1 in [Blumer *et al.*, 1989] guarantees that a hypothesis that is consistent with m labeled examples, chosen independently at random from an *arbitrary* distribution, has error smaller than ϵ with probability $1-\delta$ if

$$m \geq \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{13}{\epsilon}\right) .$$

Combining these two arguments, we get the statement of the theorem. ∎

Assuming that an initial phase of learning from unfiltered instances is used to guarantee a bound on the maximal angle between vectors, we get the following theorem.

Theorem 3.6.5: *For any $\alpha > 0$, let $\mathbf{C}_\alpha$ be the d dimensional perceptron concept class as defined in Equation (3.32), restricted to those concepts $c_{\vec{w}}$, such that $\vec{w}_0 \cdot \vec{w} > \alpha$ for some unit vector $\vec{w}_0$. Let the prior distribution over $\mathbf{C}_\alpha$ be within is within $\lambda_\mathcal{P}$ of uniform and the input distribution be within $\lambda_\mathcal{D}$ from uniform. Then the expected information gain of the queries of* **QBC** *is larger than* $0.672\alpha^{5d}\lambda_\mathcal{P}^4\lambda_\mathcal{D}$

Proof: The version space for the perceptron is a region on the d-dimensional unit sphere that is bounded by a set of great circles. We shall transform this problem into a special case of the parallel planes learning problem defined in Section 3.6.1.

Because we assume the existence of the vector $\vec{w}_0$ we can define a one-to-one mapping of the version space to a bounded convex subset of R^{d-1}. We can assume, without loss of generality, that $\vec{w}_0 = \{1, 0, \ldots, 0\}$. We can also assume that $||\vec{x}||_2 = 1$, because all instances $\vec{x}$ whose length is smaller than 1 can be mapped to $\vec{x}/||\vec{x}||_2$ without changing the label assigned to them by the concepts. The distribution over the surface of the unit sphere that is created in this way is within $\lambda_\mathcal{D}$ of uniform.

In this case the mapping of the concepts is defined by transforming the vector $\vec{w} = \{w_1, w_2, \ldots, w_d\}$ that lies on the unit sphere to the $d-1$ dimensional vector $\vec{w}' = \{w_2/w_1, w_3/w_1, \ldots, w_d/w_1\}$. The corresponding mapping of the instances maps the instance $\vec{x} = \{x_1, \ldots, x_d\}$ that lies on the unit sphere to the pair $\vec{x}' = \{x_2, \ldots, x_d\}/\sqrt{\sum_{i=2}^d x_i^2}$ and $t = -x_1//\sqrt{\sum_{i=2}^d x_i^2}$. It is easy to see that the condition that defines the perceptron $\vec{w} \cdot \vec{x} \geq 0$ is equivalent to $\vec{x}' \cdot \vec{w}' \geq t$, which is the condition that defines the corresponding parallel-plane concept.

The condition $\vec{w} \cdot \vec{w}_0 > \alpha$ is, in this case, equivalent to $w_1 > \alpha$. It is easy to check that the only examples in the transformed concept space that can be labeled both 0 and 1 by some concept in $\mathbf{C}_\alpha$ are those for which $\sqrt{\sum_{i=2}^d x_i^2} > \alpha$. This implies that the increase in the volume of an infinitesimal part of the instance space is by a factor of at most α^{-d}. Thus as the distribution over the instances on the surface of the unit sphere is within $\lambda_\mathcal{D}$ of uniform, the distribution over the transformed instance space is within $\alpha^d\lambda_\mathcal{D}$ of uniform.

To bound the distance of the prior distribution from uniform, consider the mapping of an infinitesimally small region of the version space from the sphere to the plane. Figure 3.5 illustrates this transformation for a two dimensional perceptron. This transformation maps the hyperspherical region to a larger region in the hyperplane. The factor by which the volume is increased is between 1 and α^{-d}. This can be seen by separating the transformation into two steps. In the first step, the region on the unit hypersphere is mapped to a region on a larger hypersphere. The radius of this larger hypersphere is at most α^{-1}, thus the increase in the volume is by a factor of at most $\alpha^{-(d-1)}$. In the second step, the region on the large hypersphere is mapped to the hyperplane, as the region is infinitesimally small, it can be approximated by a linear region. The increase in the volume of the region in this step is by a factor of α^{-1}. Multiplying the two factors we get α^{-d}.

As the prior distribution over the sphere is within $\lambda_\mathcal{P}$ of uniform, the distribution over the hyperplane that is generated by the mapping is within $\lambda_\mathcal{P}\alpha^d$ of uniform.

We thus have a special case of the parallel plane learning problem with close to uniform distributions. Using Theorem 3.6.3, we get the result of the theorem. ∎

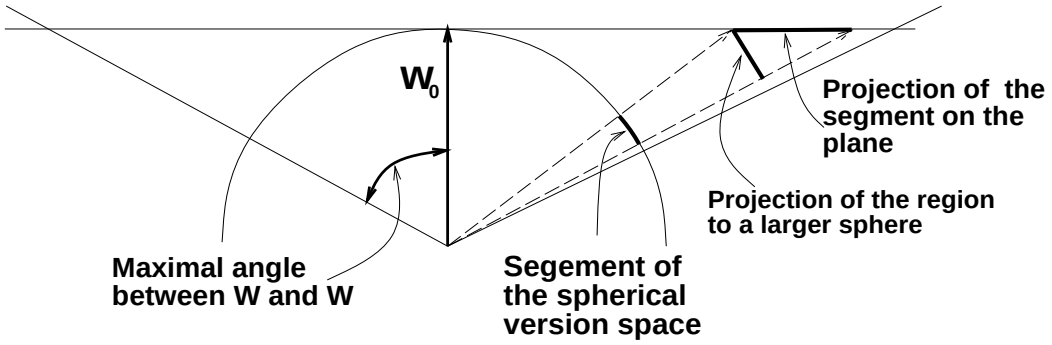


Figure 3.5: The transformation that maps the spherical version space unto the hyperplane.

3.7 Learning using unlabeled examples and membership queries

The **QBC** algorithm uses unlabeled examples in order to reduce the number of labeled examples that it needs to know. While **QBC** is a very simple algorithm it clearly does not manifest the only way of using the information provided by random unlabeled examples. In this section we define a measure of the information content of a sequence of labeled examples that is relevant directly for prediction. We show that this measure, unlike the standard measure of information that we discuss earlier in the paper, is directly related to the prediction error. We also show that is can be estimated directly when the learner has access to unlabeled random examples. From our discussion in Section 3.3, we know that this measure of information has to be sensitive to the input distribution $\mathcal{D}$, as well as to the prior $\mathcal{P}$.

The measure of information that we suggest is the expected entropy of the distribution of the label of a randomly chosen example. More specifically, if V is the version space, $\mathcal{P}$ the prior distribution, and $\mathcal{D}$ the input distribution, then we define the *label entropy* of $V \subset \mathbf{C}$ with respect to $\mathcal{P}$ and $\mathcal{D}$ as

$$\mathcal{LH}(V, \mathcal{P}, \mathcal{D}) = E_{x \in \mathcal{D}} \left(H(\Pr_{\mathcal{P}}(c(x) = 1 | c \in V)) \right) \ .$$

This measure of information can be interpreted as the expected information that we get from the label of a random example, given that we already know that it is labeled by a concept from the version space V. If the label entropy is low, then for most instances the distribution of the labels is highly biased to either 0 or 1, and thus predicting the more probable label is likely to be correct. More specifically, Haussler et. al. ([Haussler *et al.*, to appear]) have shown that the probability that the optimal Bayes prediction is incorrect can be upper and lower bounded by functions[12]of $\mathcal{LH}(V, \mathcal{P}, \mathcal{D})$:

$$H^{-1}(\mathcal{LH}(V, \mathcal{P}, \mathcal{D})) \leq$$

$$E_{x \in \mathcal{D}} \left(\min(\Pr_{\mathcal{P}}(c(x) = 1 | c \in V), 1 - \Pr_{\mathcal{P}}(c(x) = 1 | c \in V)) \right)$$

$$\leq \mathcal{LH}(V, \mathcal{P}, \mathcal{D})/2$$

Also, this measure of information is exactly the expected log-loss of the optimal prediction algorithm that outputs the probability of each label.

[12]We define $H^{-1}(x)$ as the unique $0 \leq y \leq 1/2$ such that $H(y) = x$.

While this measure is a natural measure of the information content of a sample that is relevant for prediction, it is not a useful guide for learning algorithms in the standard frameworks of distribution-free learning, because it is not clear how to estimate it when the input distribution, $\mathcal{D}$, is not known. In our model the learner can make calls to the oracles **Sample** and **Gibbs**, and in this way estimate $\mathcal{LH}(V, \mathcal{P}, \mathcal{D})$ directly. Using this estimate, the learner can search for those queries that produce the largest expected reduction in the label entropy of the version space. If the algorithm can find examples that reduce the label entropy of V by a fixed fraction at every step then we are guaranteed that the prediction error will decrease exponentially fast in the number of queries. On the other hand, note that when **QBC** decreases the prediction error below e^{-n} after n queries, then the label entropy after n queries is smaller than ne^{-n}. Thus the queries used by **QBC** in such cases decrease the label entropy of V by a fixed fraction. Thus **QBC** can be seen as an algorithm that can, for some learning problems, select queries that reduce the label entropy by a fixed fraction. While **QBC** is a very simple and surprisingly effective method of searching for such queries, it seems very possible that other algorithms for finding such queries might be more general.

The definition of label entropy can be extended to the case where the target concept is randomized. This is of special interest because we do not know how to extend **QBC** for such problems. In this case each concept $c \in \mathbf{C}$ assigns a probability to each of the two labels. We shall denote the probability assigned to the label $y \in 0, 1$ under the distribution defined by the concept c and the instance x by $\Pr_{c(x)}(y)$. In this case the regular definition of the concept space is useless, because the notion of consistency of a concept with the data is no longer deterministic. The natural generalization of the notion of the version space to this case is the Bayesian posterior distribution. The probability assigned to the concept c after observing the sample $S = \langle x_1, y_1 \rangle, \ldots, \langle x_n, y_n \rangle$ is:

$$\Pr(c|S) = \frac{\Pr(c)\,\Pr(S|c)}{\Pr(S)} = \frac{\Pr_{\mathcal{P}}(c)\ \prod_{i=1}^{n} \Pr_{c(x_i)}(y_i)}{\sum_{c' \in \mathbf{C}} \Pr_{\mathcal{P}}(c')\ \prod_{i=1}^{n} \Pr_{c'(x_i)}(y_i)}$$

and the label entropy can be defined in this case to be

$$\mathcal{LH}(V, \mathcal{P}, \mathcal{D}) = E_{x \in \mathcal{D}} \left(H \left(\sum_{c \in \mathbf{C}} \Pr(c|S) \Pr_{c(x)}(1) \right) \right) \ .$$

The relations between prediction and the information content carry on to this case. The question of finding effective algorithm for selecting queries with high information content in this case is a very interesting open question.

3.8 Summary

In this chapter we have proved that the Query by Committee algorithm is an efficient query algorithm for the perceptron concept class with distributions that are close to uniform. This establishes, by rigorous mathematical analysis, results that were found By Seung et. al. in [Seung *et al.*, 1992] using methods from statistical mechanics, which are only partially formalized and which consider only the high-dimensional limit of the problem.

We have proved that, in general, if the queries that are selected by the query by committee algorithm have high expected information gain then the prediction error is guaranteed to decrease rapidly with the number of queries. By proving that this is the case for the perceptron learning problem, we have achieved our main result.

We hope that lower bounds on the expected information gain of **QBC** can be proven for much more general learning problems. It seems that it would be very useful, in this context, to generalize Theorem 3.5.1 to allow cases in which the expected information gain is small to occur with some small probability.

The **QBC** algorithm, in its current form, does not allow for noise in the labels of the examples, and it assumes that the hypothesis space and the concept space are equal. Finding algorithms for selecting queries that work in this case is the next main step in this direction.

In this work we have explored some of the power of algorithms for learning using queries that have access to random unlabeled instances and can make membership queries. This model of learning is a natural one in contexts where unlabeled instances are much cheaper than labeled instances. An interesting general open question is how much more powerful is this model of learning from queries from the standard model for using membership queries in statistical learning.

4. Unsupervised learning of distributions on binary vectors using two layer networks

4.1 Introduction

Most of this thesis is concerned with the problem of learning mappings. The data that is supplied to the learning algorithm in this case is a set of labeled instances. As the labels are commonly seen as being supplied by some kind of a supervising teacher, this type of learning is often referred to as "supervised" learning. In this chapter we consider the problem of learning distributions. In this case the data to the learner consists just of instances. This type of learning is called "unsupervised" learning.

Suppose that we are given a large (unordered) set of binary vectors and that we wish to find the types of correlations and redundancies that exist between the bits in these vectors. We assume that each binary vector is of the form $\vec{x} = \{x_1, \ldots, x_n\} \in \{\pm 1\}^n$, and that each vector is generated independently at random according some unknown distribution on $\{\pm 1\}^n$. Such an assumption is natural, for instance, when each instance consists of (possibly noisy) measurements of n different binary attributes of a randomly selected object. Our interest is in cases where the dimension n of the vectors is large, say $n > 50$. One example of this type of scenario is when the instances are binary images of handwritten digits, where each bit corresponds to the black or white color of a single pixel in the image. The correlations that we expect to see in this case correspond to the fact that the values of neighboring pixels or pixels that lie along lines or curves are strongly dependent on each other.

Knowledge of the correlations between different bits of the binary vector is useful when we want to use a set of measurements for various classification and prediction tasks. The idea that features that are useful for classification can be deduced from the distribution of typical inputs is the basis of several existing algorithms for unsupervised learning. One type of algorithm selects projections of the input based on Principle Component analysis [Sanger, 1989, Oja, 1989]. Another type of algorithm clusters data based on an assumption that the underlying distribution is a mixture of Gaussians [Everitt and Hand, 1981, Nowlan, 1990]. The combination model presented in this paper is related to both of these lines of work and has some advantages over each of them.

If we find a good model of the distribution, we can tackle other interesting learning problems, such as the problem of estimating the conditional distribution on certain components of the vector $\vec{x}$ when provided with the values for the other components (a kind of regression problem), or predicting the actual values for certain components of $\vec{x}$ based on the values of the other components (a kind of pattern completion task). In the example of the binary images presented above, this would amount to the task of recovering the value of a pixel whose value has been corrupted. We can often also use the distribution model to help us in a supervised learning task. This is because it is often easier to express the mapping of an instance to the correct label by using "features" that are correlation patterns among the bits of the instance. For example, it is easier to describe each of the ten digits in terms of patterns such as lines and circles, rather than in terms of the values of individual pixels, that are more likely to change between different instances of the same digit.

The process of learning an unknown distribution from examples is usually called *density estimation* or *parameter estimation* in statistics, depending on the nature of the class of distributions used as models. There has been considerable work on density/parameter estimation for distributions on real vector spaces (see e.g. [Duda and Hart, 1973b]), and less on binary vector spaces. The most popular mainstream statistics models for distributions on $\{\pm 1\}^n$ for large n appear to be small mixtures of Bernoulli product distributions[1] [Everitt and Hand, 1981, Nowlan, 1990], and models in which only k-wise dependencies between the components of the input are allowed, for some $k << n$ [Freeman, 1987, Cox and Snell, 1989]. Newer and more exciting models include Bayes networks [Pearl, 1988] and Markov random fields [Pearl, 1988, Geman and Geman, 1984, Geman, 1986]. In the neural network area, both Hopfield nets [Hopfield, 1982] and Boltzmann machines [Ackley *et al.*, 1985] can be used as models of probability distributions on $\{\pm 1\}^n$ for relatively large n. We will look at a class of models defined by a special type of Boltzmann machine.

Hopfield networks, Boltzmann machines and Markov random fields are all based on the statistical physics concepts of energy and local interaction between units whose state is binary.[2] The models defined by Hopfield networks and Boltzmann Machines are special cases of the more general Markov random field model. The units in a Hopfield network correspond to the bits of the binary vectors and the interaction between units are restricted to symmetric pairwise interactions. Boltzmann machines also employ only pairwise interactions, but in addition to the units that correspond to bits of the data vectors, commonly called the *input* units, there are *hidden* units, which correspond to unobserved binary variables. These hidden units interact with the input units and generate correlations between the vector bits that the input units represent. The distribution of the binary vectors generated by the Boltzmann Machine is defined as the marginal distribution induced on the state of the input units by the Markov random field over all units, both observed and hidden.

While the Hopfield network is relatively well understood, it is limited in the types of distributions that it can model. On the other hand, Boltzmann machines are universal in the sense that they are powerful enough to model any distribution (to any degree of approximation), but the mathematical analysis of their capabilities is often intractable. Moreover, the standard learning algorithm for the Boltzmann machine, a gradient ascent heuristic to compute the maximum likelihood estimates for the weights and thresholds, requires repeated stochastic approximation, which results in unacceptably slow learning. Many methods have been proposed to speed up learning in Boltzmann machines. One of these methods is the mean-field approximation [Peterson and Anderson, 1987]. In Section 4.2.2 we shall see some relations between one of our learning rules and the mean field approximation.

---

[1] A Bernoulli product distribution is a distribution over binary vectors in which each component is chosen independently of the rest.

[2] Informally, a Markov random field consists of a set of random variables that are connected as nodes in a graph. The distribution of each random variable is determined by the value of its neighbors. In other words, given the value of all the neighbors of random variables, the value of the random variable is independent of the state of the rest of the random variables. The Markov process is a special case of the Markov field in which each random variable corresponds to a specific time step and its neighbors are the random variables that correspond to the previous and the succeeding time steps. In general, Markov-field distributions have a canonical description that is based on the concept of interaction energy.

In our research we have attempted to narrow the gap between Hopfield networks and Boltzmann machines by finding a model that will be powerful enough to be universal, yet simple enough to be analyzable and computationally efficient.[3] The model that we use in this work is essentially a Boltzmann Machine whose connection graph is bipartite. There are two types of nodes: "input" nodes and "hidden" nodes. Each input node is connected to each of the hidden nodes, and no other connections exist. We call this model the *influence combination machine*, or, for short, the combination machine. We refer to the distribution that is defined on the binary vectors by the combination machine as the *combination model*. This type of Boltzmann machine was previously studied by Smolensky in his *harmony theory* [Rumelhart and McClelland, 1986][Ch.6]. In his work he discusses several possible ways of using this type of model for solving problems in Artificial Intelligence. In our work we concentrate on the mathematical properties of the model and on efficient algorithms for learning the model from random instances.

The combination machine consists of two types of units: input units, each of which holds one component of the input vector, and hidden units, representing hidden variables. There is a weighted connection between each input unit and each hidden unit, and no connections between input units or between hidden units (see Figure 4.1). The presence of the hidden units induces dependencies, or correlations, between the variables modeled by input units. To illustrate the representational power of the combination model, consider the distribution of people that visit a specific coffee shop on Sunday. Let each of the n input variables represent the presence $(+1)$ or absence (-1) of a particular person that Sunday. These random variables are clearly not independent. For example, if Fred's wife and daughter are there, it is more likely that Fred is there as well, if you see three members of the golf club, you expect to see other members of the golf club, if Bill is there, you are unlikely to see his ex-wife Brenda there, etc. This situation can be modeled by a combination model in which each hidden variable represents the presence or absence of a social group. The weights connecting a hidden unit and an input unit measure the tendency of the corresponding person to be associated with the corresponding group. In this coffee shop situation, several social groups may be present at the same time, exerting a combined influence on the distribution of customers. In Sections 4.2.3 and 4.2.4 we discuss why the combination model is better for describing this type of distributions than popular models such as the mixture model and principal components methods.[4]

We show that the combination model is a universal model in the sense that any probability distribution on $\{\pm 1\}^n$ can be represented by a combination model with n input units to within any desired accuracy. Then we show that the standard Boltzmann machine learning rule, when applied to a combination model, can be computed in closed form, instead of using random sampling techniques. Thus we get a faster learning algorithm than the standard Boltzmann rule that is also exact. The computational complexity of

[3] Recent work on modeling correlations by hidden units has also been done by Radford M. Neal [Neal, 1990]. In his work he gives a different variant of the Boltzmann Machine algorithm that uses distribution models similar to Judea Pearl's Bayes Networks [Pearl, 1988, Gefner and Pearl, 1987]. His model is superior to the Boltzmann Machine in the sense that the connection weights are interpreted as conditional probabilities, which is a more accessible interpretation than local energy interactions. The learning algorithms that Neal used are based on stochastic approximation. The question of whether a two-layer model of this type has universal representation capabilities is open.

[4] Noisy-OR gates have been introduced in the framework of Bayes Networks to allow for such combinations [Pearl, 1988]. However, using this in networks with hidden units has not been studied, to the best of our knowledge.

the learning rule is exponential in the number of hidden units. However, under certain natural conditions we show that there exists a good approximation that requires only polynomial time.

We then explore the relationships between the distributions generated by the combination model and those studied in Projection Pursuit density estimation [Huber, 1985, Friedman *et al.*, 1984, Friedman, 1987]. We show that the search for hidden variables that have a strong influence on the input distribution can be interpreted as a search for projections of the input that have a non-Normal marginal distribution. Based on this observation, we propose a learning algorithm based on exploratory projection pursuit for the combination model. This method is a greedy method that adds a single hidden unit at a time to the model. The time complexity of this method is linear in the number of hidden units compared to the exponential complexity of the gradient based method. However, while the gradient based method is guaranteed to converge to a local minimum in the model space, the projection pursuit method does not have this guarantee.

We conclude this paper with results of some experiments. The first set of experiments compare the two learning algorithms on synthetically generated data, and demonstrate their advantages and deficiencies. The second set of experiments compare the performance of the combination model to that of the mixture model and demonstrate the difference in the type of distribution representations that they generate.

4.2 The influence combination distribution model

4.2.1 Notation

For the most part, we use standard algebraic notation in our formulas. Elements from the n-dimensional spaces R^n and $\{-1,+1\}^n$ are denoted by vectors $\vec{x},\vec{y},\ldots$. We denote by $||\vec{x}||_1$,$||\vec{x}||_2$ the l_1 and l_2 norms of $\vec{x}$, and by $\vec{x}\cdot\vec{y}$ the dot product between two vectors. We use the standard hyperbolic trigonometric functions

$$\sinh(x)=\frac{e^x-e^{-x}}{2},\ \ \cosh(x)=\frac{e^x+e^{-x}}{2},\ \ \tanh(x)=\frac{\sinh(x)}{\cosh(x)}\ .$$

We denote the natural base logarithm by "ln". Finally, we use the function $\mathrm{logistic}(x)=1/(1+\exp(-x))$ that is commonly used in the definition of Boltzmann Machines.

4.2.2 The Model

In this section we present the combination machine and the corresponding distribution model, which is the influence combination distribution model. The combination machine is a simple Connectionist type model which is a special case of the Boltzmann Machine [Ackley *et al.*, 1985]. As we shall see, the simplicity of this special case makes it easier to analyze than the general Boltzmann machine and allows the use of more efficient learning algorithms. At the same time, the model is still powerful enough to approximate any distribution of binary vectors.

To model a distribution on $\{\pm 1\}^n$ we use a machine with $n+m$ units. There are two types of units, n *input* units, each of which represents a single bit in the random vector, and m *hidden* units, whose role

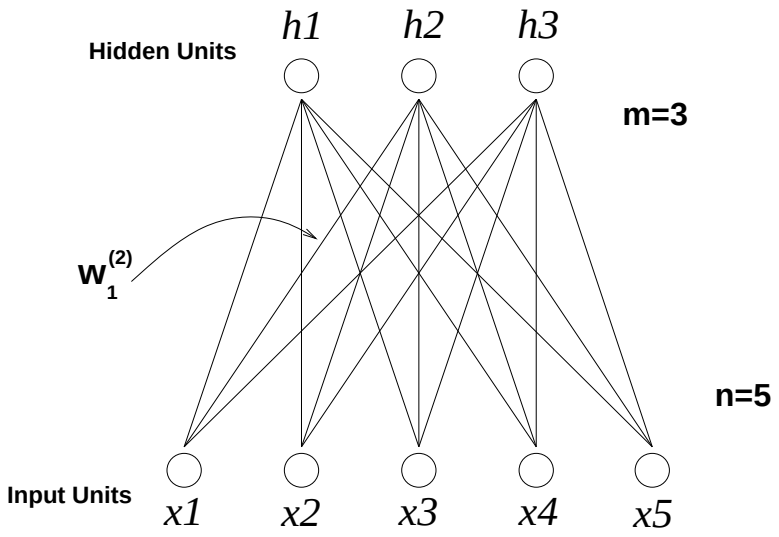


Figure 4.1: The bipartite graph of the combination model

is the create correlation between the values of the input units. These units are connected in a bipartite graph as illustrated in Figure (4.1).

The random variables represented by the input units each take values in $\{+1, -1\}$, while the hidden variables, represented by the hidden units, take values in $\{0, 1\}$. The *state* of the machine is defined by the values of these random variables. We denote by $\vec{x} = (x_1, \ldots, x_n) \in \{\pm 1\}^n$ the state of the input units, and by $\vec{h} = (h_1, \ldots, h_m) \in \{0, 1\}^m$ the state of the hidden units.

There are $m(n+1)$ real-valued parameters associated with the machine. Each particular setting of these parameters defines the *parameter vector* of the machine. Each parameter vector defines a distribution on the states of the machine. Summing over the state of the hidden units we get a distribution on the input units, which is the influence combination distribution defined by the particular parameter vector. There are two variants of the combination model, which we call the binary valued and the real valued combination machines. While we are mostly interested in the binary model, the real valued model is a useful approximation in some cases.

The parameters are all real-valued and are defined as follows. There is a *weight* parameter associated with each edge in the bipartite graph. We denote by $\omega_j^{(i)}$ the weight of the edge connecting the ith hidden unit to the jth input unit. We also use $\vec{\omega}^{(i)}$ to denote the vector of all n weights associated with the ith hidden unit.[5] There is a *bias* parameter associated with each hidden unit. We denote the bias of the ith hidden unit by $\theta^{(i)} \in R$. The complete parameter vector of a binary combination model is denoted by $\phi_B = \{(\vec{\omega}^{(1)}, \theta^{(1)}), \ldots, (\vec{\omega}^{(m)}, \theta^{(m)})\}$. For a given ϕ_B, the *energy* of a state of the combination machine is defined as

$$E(\vec{x}, \vec{h} | \phi_B) = -\sum_{i=1}^{m} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i \tag{4.1}$$

and the probability of a state is defined to be

[5] In [Rumelhart and McClelland, 1986][Ch.6], binary connection weights are used, here we use real-valued weights.

$$Pr(\vec{x},\vec{h}|\phi_B) = \frac{1}{Z_B} e^{-E(\vec{x},\vec{h}|\phi_B)} \quad \text{where} \quad Z_B = \sum_{\vec{x},\vec{h}} e^{-E(\vec{x},\vec{h}|\phi_B)}.$$

We find it useful to define the "combined weight" of a particular state of the hidden units as the sum of the weight vectors corresponding to the hidden units whose state is 1:

$$\vec{\omega}(\vec{h}) = \sum_{i=1}^{m} h_i \vec{\omega}^{(i)}$$

Plugging the definition of the energy into the definition of Z_B, we get that

$$Z_B = \sum_{\vec{x},\vec{h}} \exp\left(\sum_{i=1}^{m} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right) .$$

Expanding the sum in the exponent we get that

$$Z_B = \sum_{\vec{h}\in\{0,1\}^m} \left(\exp(\sum_{i=1}^{m} h_i \theta^{(i)}) \sum_{\vec{x}\in\{-1,+1\}^n} \exp(\vec{x} \cdot \vec{\omega}(\vec{h}))\right) .$$

Expanding the sum over $\vec{x} \in \{-1,+1\}^n$, we get that

$$Z_B = \sum_{\vec{h}\in\{0,1\}^m} \left(\exp(\sum_{i=1}^{m} h_i \theta^{(i)}) \prod_{j=1}^{n} (\exp(\vec{\omega}(\vec{h})_j) + \exp(-\vec{\omega}(\vec{h})_j))\right) ,$$

where $\vec{\omega}(\vec{h})_j$ denotes the jth component of $\vec{\omega}(\vec{h})$. Using the definition of $\cosh(x)$, we can rewrite the last expression as

$$Z_B = 2^n \sum_{\vec{h}\in\{0,1\}^m} \left[\exp(\sum_{i=1}^{m} h_i \theta^{(i)}) \prod_{j=1}^{n} \cosh(\vec{\omega}(\vec{h})_j)\right] . \tag{4.2}$$

Note that the trivial model, in which there are no hidden units, defines the uniform distribution over the state vectors $\vec{x}$. In the general case the probability distribution over possible state vectors on the input units is given by

$$Pr(\vec{x}|\phi_B) = \sum_{\vec{h}\in\{0,1\}^m} Pr(\vec{x},\vec{h}|\phi_B) = \frac{1}{Z_B} \sum_{\vec{h}\in\{0,1\}^m} \exp\left(\sum_{i=1}^{m} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right) \tag{4.3}$$

By separating the sum over $\vec{h}$ into a sum over all $\vec{h}$ such that $h_m = 0$ and a sum over all $\vec{h}$ such that $h_m = 1$, we can rewrite Equation (4.3) in the following form:

$$Pr(\vec{x}|\phi_B) = \frac{1}{Z_B}\left(e^0 + e^{\vec{\omega}^{(m)}\cdot\vec{x}+\theta^{(m)}}\right) \sum_{\{h_1,\ldots,h_{m-1}\}\in\{0,1\}^{m-1}} \exp\left(\sum_{i=1}^{m-1} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right)$$

Repeating this manipulation for all m components of $\vec{h}$ we get that

$$Pr(\vec{x}|\phi_B) = \frac{1}{Z_B} \prod_{i=1}^{m} \left(1 + e^{\vec{\omega}^{(i)}\cdot\vec{x}+\theta^{(i)}}\right) . \tag{4.4}$$

Equation (4.4) is a simple closed form representation of the distribution defined by the parameter vector ϕ_B. Notice that the hidden unit variables, h_i, are not explicitly present in this formula. Each factor in the product is associated with one hidden unit in the corresponding machine. This product form is particular to the combination model, and does not hold for general Boltzmann machines. Product form distribution models have been used for density estimation in Projection Pursuit [Huber, 1985, Friedman *et al.*, 1984, Friedman, 1987]. We shall look further into this relationship in Section 4.3.3.

In some of the following discussion we shall find it useful to use a variant of the combination model that defines distributions over the whole real space R^n, i.e. to allow each input to have any real-value instead of limiting it to only +1 and -1. The structure of the machine is the same, we keep the hidden variables $\{0,1\}$-valued, and the distribution is defined in a similar way, but the energy function has an extra term that is necessary for ensuring that the distribution can be normalized. This term corresponds to each input unit having a connection of strength -1 to itself. To differentiate between the binary and the real-valued models we subscript quantities relating to the real-valued model by R. The energy of a particular state of the real-valued model is given by

$$E(\vec{x},\vec{h}|\phi_R) = -\left(\sum_{i=1}^{m}(\vec{w}i\cdot\vec{x}+\theta^{(i)})h_i\right)+\frac{1}{2}||\vec{x}||_2^2, \tag{4.5}$$

which produces the following distribution over the R^n:

$$Pr(\vec{x}|\phi_R) = \sum_{\vec{h}\in\{0,1\}^m} Pr(\vec{x},\vec{h}|\phi_R) = e^{-\frac{1}{2}||\vec{x}||_2^2}\frac{1}{Z_R}\prod_{i=1}^{m}\left(1+e^{\vec{w}i\cdot\vec{x}+\theta^{(i)}}\right), \tag{4.6}$$

where

$$Z_R = \int_{R^n}\sum_{\vec{h}\in\{0,1\}^m}\exp\left(-E(\vec{x},\vec{h}|\phi_R)\right)d\vec{x} \tag{4.7}$$

$$= \sum_{\vec{h}\in\{0,1\}^m}\int_{R^n}\exp\left(-\frac{1}{2}||\vec{x}||_2^2+\sum_{i=1}^{m}(\vec{w}i\cdot\vec{x}+\theta^{(i)})h_i\right)d\vec{x}$$

$$= (2\pi)^{n/2}\sum_{\vec{h}\in\{0,1\}^m}\exp\left[\sum_{i=1}^{m}h_i\theta^{(i)}+\frac{1}{2}||\vec{\omega}(\vec{h})||_2^2\right], \tag{4.8}$$

using the integral of the Gaussian distribution.

We discuss the relation between the real-valued and the binary-valued model in Section 4.2.6.

4.2.3 Discussion of the model

The right hand side of Equation (4.4) has a simple intuitive interpretation. The ith factor in the product corresponds to the hidden variable h_i and is an increasing function of the dot product between $\vec{x}$ and the weight vector of the ith hidden unit. Hence an input vector $\vec{x}$ will tend to have large probability when it is in the direction of one of the weight vectors $\vec{w}i$ (i.e. when $\vec{w}i\cdot\vec{x}$ is large), and small probability otherwise. This is the way that the hidden variables can be seen to exert their "influence"; each corresponds to a preferred or "prototypical" direction in space. The bias parameter $\theta^{(i)}$, together with the length $||\vec{w}i||_2$

of the weight vector, control the strength of the influence of the ith hidden variable in comparison with the other hidden variables, as well as its "width", i.e. how close in direction $\vec{x}$ has to be to $\vec{w}i$ before it significantly influences its probability. Increasing either $||\vec{w}i||_2$ or $\theta^{(i)}$ increases the strength of the influence of the hidden unit. Decreasing $\theta^{(i)}$ and, at the same time, increasing $||\vec{w}i||_2$, decreases the "width" of the influence, making the influence of the ith hidden unit more restricted to input vectors whose direction is very close to the direction of $\vec{w}i$. This is true for both the binary-valued and the real-valued combination models.

Equation (4.3) shows that the combination model can be written as a mixture of 2^m distributions of the form

$$\frac{1}{Z(\vec{h})} \exp\left(\sum_{i=1}^{m} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right),$$

where $\vec{h} \in \{0,1\}^m$ and $Z(\vec{h})$ is the appropriate normalization factor. Each of these distributions is a product of n Bernoulli distribution, i.e. the x_j is drawn independently at random and attains a value of -1 or $+1$ with probabilities logistic$(-2\vec{\omega}(\vec{h})_j)$ and logistic$(+2\vec{\omega}(\vec{h})_j)$ respectively, which implies that the mean of x_j in according to this distribution is $\tanh(\vec{\omega}(\vec{h})_j)$. We shall refer to this type of distribution as a "Bernoulli product distribution". The combination model is a mixture of 2^m Bernoulli product distributions, each corresponding to a setting of $\vec{h}$ and each having a mixture coefficient $Z(\vec{h})$.

It is interesting to compare the class of combination models to the standard class of models defined by a mixture of Bernoulli product distributions. The same bipartite graph described in Figure (4.1) can be used to define a standard mixture model. Assign each of the m hidden units a weight vector $\vec{w}i$ and a probability p_i such that $\sum_{i=1}^{m} p_i = 1$. To generate an example, choose *one* of the hidden units according to the distribution defined by the p_i's, and then choose the vector $\vec{x}$ according to $P_i(\vec{x}) = \frac{1}{Z_i} e^{\vec{w}i \cdot \vec{x}}$, where Z_i is the appropriate normalization factor so that $\sum_{\vec{x} \in \{\pm 1\}^n} P_i(\vec{x}) = 1$. We thus get the distribution

$$P(\vec{x}) = \sum_{i=1}^{m} \frac{p_i}{Z_i} e^{\vec{w}i \cdot \vec{x}} \tag{4.9}$$

This form for presenting the standard mixture model emphasizes the similarity between this model and the combination model. A vector $\vec{x}$ will have large probability if the dot product $\vec{w}i \cdot \vec{x}$ is large for some $1 \leq i \leq m$ (so long as p_i is not too small). However, unlike the standard mixture model, the combination model allows more than one hidden variable to be $+1$ for any generated example. This means that several hidden influences can combine in the generation of a single example, because several hidden variables can be $+1$ at the same time.

To see why this is useful, consider two examples. First, consider the coffee shop example given in the introduction. At any moment of time it is reasonable to find *several* social groups of people sitting in the shop. The combination model will have a natural representation for this situation, while in order for the standard mixture model to describe it accurately, a hidden variable has to be assigned to each combination of social groups that is likely to be found in the shop at the same time. Similarly, when we want to represent the distribution of binary images of digits, it is reasonable to assume that each specific image contains *several* patterns, such as lines and curves. Of course, the whole digit can be perceived as a pattern, in which case the mixture model is the relevant distribution model. However, we claim that

it is often more appropriate to represent each digit image as a combination of patterns rather than a single pattern. In other words, we claim that, for typical sets of images of digits, the maximal likelihood combination model will have larger likelihood than a mixture model with the same number of parameters. In Section 4.4 we give experimental evidence to support this claim. In cases where this claim is correct the combination model is exponentially more succinct than the standard mixture model, and naturally captures the underlying product structure of the distribution. Of course, if the space of hidden variables does not have a product structure of this type, then the combination model is no better than the standard mixture model.

In analogy with the binary combination model, the real-valued combination model can be shown to represent a mixture of 2^m symmetric Gaussian distributions. From Equation (4.6) we get that for the empty case, $m = 0$, where there are no hidden variables present, the distribution is a symmetric Gaussian by definition. When a single hidden variable is present, the distribution becomes

$$Pr(\vec{x}|\phi_R) = e^{-\frac{1}{2}||\vec{x}||_2^2}\frac{1}{Z}\left(1 + e^{\vec{w}1\cdot\vec{x}+\theta^{(1)}}\right) = \frac{1}{Z}\left(e^{-\frac{1}{2}||\vec{x}||_2^2} + e^{-\frac{1}{2}||\vec{x}||_2^2+\vec{w}1\cdot\vec{x}+\theta^{(1)}}\right) =$$

$$\frac{1}{Z}\left(e^{-\frac{1}{2}||\vec{x}||_2^2} + e^{-\frac{1}{2}||\vec{x}-\vec{w}1||_2^2+\frac{1}{2}||\vec{w}1||_2^2+\theta^{(1)}}\right).$$

This is a mixture of two Gaussians, both of which have spherical symmetry. They differ only in the location of the mean, which is $\vec{0}$ for the first component and $\vec{w}1$ for the second component, and in their relative probabilities (mixture weights). Each additional hidden unit has the effect of transforming the previous distribution into a mixture of two distributions, one is the previous distribution, and the other is the original distribution shifted by $\vec{w}i$ (See Figure 4.2).

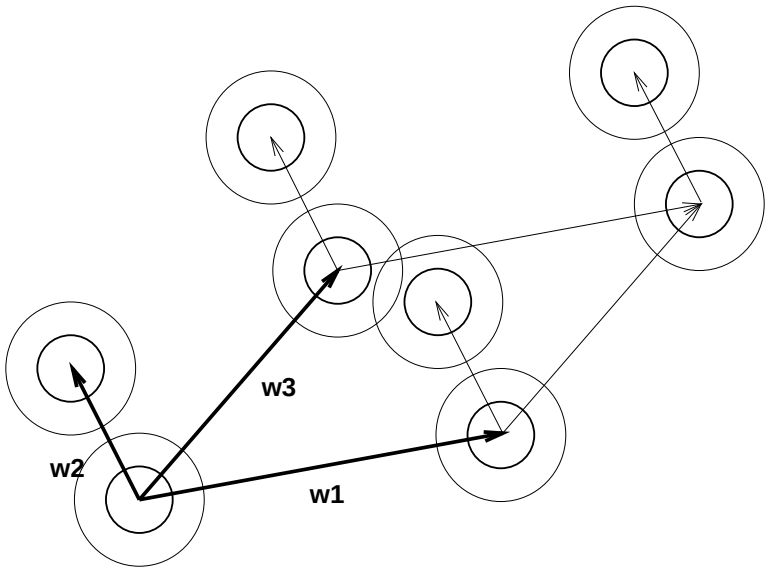


Figure 4.2: The distribution over R^2 generated by a (real valued) combination model with three hidden units. Each pair of concentric circles denotes a single Gaussian distribution. The distribution defined by the combination model is a mixture of these eight Gaussians.

In the general case the combination model with m hidden units is equivalent to a mixture of 2^m Gaussians whose expected values are located at the combined weight, $\vec{\omega}(\vec{h})$, corresponding to each of the 2^m possible states of $\vec{h}$.[6] This interpretation of the real-valued model will be used in Section (4.3.6) in a Projection Pursuit algorithm for learning the combination model.

4.2.4 Comparison with principal components analysis

Principal Component Analysis (PCA) is a popular method for the analysis of high order correlations (see e.g. [Jolliffe, 1986]). Many algorithms for unsupervised learning are based on this method, among them some learning rules for neural networks [Sanger, 1989, Oja, 1989]. The method is based on the covariance matrix, which measures pairwise correlations among input bits. The main assumption underlying the method is that the low dimension projections of the data that retain the largest amount of information are those projections that have the largest variance. One justification of this assumption is that if the data has a simple enough distribution such as a Gaussian distribution then the reconstruction of the original input from its projections is optimal for this choice of projections. The directions with largest variance are equal to the directions of the eigenvectors of the covariance matrix that have the largest eigenvalues.

The neural network implementation of PCA is usually a two layer network with the same architecture as the combination model. The learning rule, however, is different, and tries to make the weight vectors of the hidden units equal to eigenvectors of the covariance matrix of the input. The outputs of the hidden units are thus projections of the data (or a nonlinear transformation of such projections).

This type of network is capable of representing each input as a combination of correlation patterns. In this sense it is as powerful as the combination model and does not suffer from the deficiencies of mixture models described in the previous section. However, as this method of analysis is based only on the second order correlations among pixels it necessarily ignores part of the structure of the distribution. In the combination model, on the other hand, each hidden unit can represent correlations of arbitrary order. We claim that some natural distributions have strong high order correlation and that taking into account only the second order correlations ignores some of the most important information available in the distribution. In Section 4.4 we shall give some experimental evidence to support this claim.

4.2.5 Universality of the model

Despite its limited connectivity, it is not hard to show that the class of binary combination models is universal in the sense that for every n and every distribution on $\{\pm 1\}^n$ there is a combination model with n input units that approximates that distribution to within any desired accuracy. The argument is similar to an argument for the same claim regarding the class of mixtures of Bernoulli product distributions.

Assume first that the distribution we want to estimate is $Pr(\vec{x}) = p$ for $\vec{x} = (1, 1, \ldots, 1)$ and $Pr(\vec{x}) = \frac{1-p}{2^n-1}$ for $\vec{x} \neq (1, 1, \ldots, 1)$. Here we need only one hidden unit. We define $q = \frac{p(2^n-1)}{1-p}$ and choose $\vec{\omega}^{(1)} = (a, a, \ldots, a)$ and $\theta^{(1)} = -na + \ln(q-1)$, where $a = \frac{1}{2}\ln(q-1) + \frac{1}{2}\ln(1/\epsilon)$. We get the following values for $f(\vec{x}) := 1 + e^{\vec{\omega}^{(1)} \cdot \vec{x} + \theta^{(1)}}$.

[6] Compare this to the mixture of Bernoulli products whose expected values are $\tanh(\vec{\omega}(\vec{h}))$. A more detailed comparison of the two models will be given in Section 4.2.6.

If $\vec{x} = (+1, +1, \ldots, +1)$, then $f(\vec{x})$ is equal to q. For a vector where exactly one component is equal to -1 and all the rest are $+1$, $f(\vec{x})$ is equal to $1 + \epsilon$, and for a vector $\vec{x}$ which has k components that are equal to -1, $f(\vec{x})$ is equal to $1 + (q-1)(\epsilon/(q-1))^k \leq 1 + \epsilon^k$. By setting ϵ small enough we can make $1 + e^{\vec{\omega}^{(1)} \cdot \vec{x} + \theta^{(1)}}$ arbitrarily close to 1 for all $\vec{x} \neq \vec{x} = (+1, +1, \ldots, +1)$. Normalizing the distribution to sum to 1 we can get a distribution that is arbitrarily close to the desired distribution.

To approximate an arbitrary distribution, we multiply 2^n factors, each approximating a distribution that is highly concentrated on a single setting of $\vec{x}$ and almost uniform on all other settings. By appropriate choice of the parameters we can approximate the arbitrary distribution closely for each value of $\vec{x}$. Of course this requires exponentially many hidden units, but this is unavoidable since it requires an exponential number of parameters to specify an arbitrary distribution over $\{\pm 1\}^n$ in any reasonable parametric model.

Of course, we are interested in cases where the distribution of the data can be represented well by a *small* combination model. While a general distribution might require many hidden units to model it, distributions that are encountered in nature are often simple, and can be modeled well by a model that has only a small number of hidden units. In Section 4.4 we show that the distribution of images of handwritten digits can be approximated well by a combination model with few hidden units.

4.2.6 Relations between the binary-valued and the real-valued models

Two variants of the combination model were introduced in Section 4.2.2, the binary-valued model (Equations (4.1) to (4.4)) and the real-valued model (Equations (4.5) to (4.8)). The binary-valued model is the natural model for representing distributions of binary vectors, and thus, ideally, we would like to use only this model. On the other hand, the real-valued model has properties that make it possible to use more efficient learning algorithms to learn it. As we show in this section, the real-valued model is an approximation of the binary model when the weights are all small. Thus we can use the algorithms for the real-valued model to find an approximate parameter vector of the binary model.

The real-valued model defines a density function, in contrast with the binary model, which defines a point mass distribution. However, the ratio between the densities assigned by a real-valued model to any pair of points in $\{-1, +1\}^n$ is equal to the ratio of the probabilities assigned to the same points by a binary model with the same parameters. This is because the factor of $e^{-\frac{1}{2}||\vec{x}||_2^2}$ in the density function is equal to $e^{-n/2}$ for all vectors in $\{-1, +1\}^n$.

This does not mean that the maximum-likelihood parameter vector for a given set of examples is equal for both models. This is because the normalization factors Z_B and Z_R are different for each of the two cases. However, as we shall now see, when the weight vectors $\vec{\omega}$ are small the normalization factors are very close to each other.

Recall Equation (4.2):

$$Z_B = 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^{m} h_i \theta^{(i)} \right) \prod_{j=1}^{n} \cosh\left(\vec{\omega}(\vec{h})_j \right) \right] .$$

The Taylor expansion of $\cosh(x)$ around $x = 0$ is:

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots$$

thus the first order approximation of Z_B for small values of $\omega_j^{(i)}$ is:

$$Z_B \approx 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^{m} h_i \theta^{(i)} \right) \prod_{j=1}^{n} \left(1 + \frac{1}{2} \left(\vec{\omega}(\vec{h})_j \right)^2 \right) \right]$$

On the other hand, note that Equation (4.8) can also be written as:

$$Z_R = (2\pi)^{n/2} \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^{m} h_i \theta^{(i)} \right) \prod_{j=1}^{n} \exp\left(\frac{1}{2} \left(\vec{\omega}(\vec{h})_j \right)^2 \right) \right] ,$$

The Taylor expansion of $\exp(x)$ around $x = 0$ is

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

thus the first order approximation of Z_R for small values of $\omega_j^{(i)}$ is:

$$Z_R \approx (2\pi)^{n/2} \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^{m} h_i \theta^{(i)} \right) \prod_{j=1}^{n} \left(1 + \frac{1}{2} \left(\vec{\omega}(\vec{h})_j \right)^2 \right) \right] \approx \left(\frac{\pi}{2} \right)^{n/2} Z_B \ .$$

The fact that the two models differ by a constant factor is of no consequence when looking for the maximal-likelihood parameter vector, because this constant factor disappears in the derivative of the log of the likelihood.

The difference between the two approximations is of the order of $||\vec{\omega}(\vec{h})||_2^4$. Thus if $||\vec{\omega}(\vec{h})||_2$ is much smaller than 1 the approximation is reasonable.

There is another way in which the two models can be compared. In Section 4.2.3 we have shown that both the real-valued and the binary-valued combination models are equivalent to mixture models. The real-valued model is equal to a mixture of 2^m Gaussian distributions. Each mixture component corresponds to a setting of $\vec{h}$ and has an expected value of $\vec{\omega}(\vec{h})$. Similarly, the binary-valued combination model is equivalent to a mixture of 2^m Bernoulli product distributions, each of which has an expected value of $\tanh(\vec{\omega}(\vec{h}))$. When $\vec{\omega}(\vec{h})$ is small $\tanh(\vec{\omega}(\vec{h})) \approx \vec{\omega}(\vec{h})$.

It is easy to show that every one dimensional projection of a Gaussian distribution generates a Normal marginal distribution. Thus the marginal distribution that is generated by the real-valued combination model is a mixture of normal distributions. Diaconis and Friedman [Diaconis and Freedman, 1984] have shown that, in some sense, *most* "well-behaved" distributions generate a marginal distribution that is close to normal when projected on a randomly chosen direction. In particular, the uniform distribution on the 2^n binary vectors in $\{-1, +1\}^n$ generates, with very high probability, a marginal distribution that is close to the normal distribution, when the projection direction is chosen uniformly at random from the n dimensional sphere, and n is large. In Appendix B, we show that this is also true for Bernoulli product distributions, if the distributions of the individual coordinates are not too biased. Thus, under reasonable assumptions, the marginal distribution that is generated by the binary valued combination model is also mixture of normal distributions.

In addition, if the weight vectors, $\vec{w}i$, are short, then the mixture coefficients and the means of the mixture components of the two models are close, which implies that the projections of the distributions defined by the real-valued model and the binary valued model with the same parameters are very close to each other. We use this correspondence in our analysis of the projection pursuit learning methods, which are based on properties of projections of the data.

4.3 Learning the model from examples

4.3.1 Learning by gradient ascent on the log-likelihood

We now suppose that we are given a sample consisting of a set S of vectors in $\{\pm 1\}^n$ drawn independently at random from some unknown distribution. Our goal is to use the sample S to find a good model for this unknown distribution using a combination model with m hidden units, if possible. The method we investigate here is the method of maximum likelihood estimation using gradient ascent. The goal of learning is reduced to finding the set of parameters for the combination model that maximizes the (log of the) probability of the set of examples S. In fact, this gives the standard learning algorithm for general Boltzmann machines [Ackley *et al.*, 1985]. For a general Boltzmann machine this would require stochastic estimation of the parameters. As stochastic estimation is very time-consuming, the result is that learning is very slow. In this section we show that stochastic estimation need not be used for the combination model.

From Equation (4.4), the log of the likelihood of a sample of input vectors $S = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(N)}\}$, given a particular setting $\phi_B = \{(\vec{\omega}^{(1)}, \theta^{(1)}), \ldots, (\vec{\omega}^{(m)}, \theta^{(m)})\}$ of the parameters of the model is:

$$\text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} \ln Pr(\vec{x}|\phi_B) = \sum_{i=1}^{m} \left(\sum_{\vec{x} \in S} \ln(1 + e^{\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}}) \right) - N \ln Z_B \ . \tag{4.10}$$

Taking the gradient of the log-likelihood results in the following formulas. For the bias parameters we get:

$$\frac{\partial}{\partial \theta^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{x} \in \{\pm 1\}^n} Pr(\vec{x}|\phi_B) \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} \tag{4.11}$$

and for the jth component of $\vec{\omega}^{(i)}$

$$\frac{\partial}{\partial \omega_j^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{x} \in \{\pm 1\}^n} Pr(\vec{x}|\phi_B) x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} \tag{4.12}$$

The purpose of the clamped and unclamped phases (also called action and sleep phases) in the Boltzmann machine learning algorithm is to approximate these two expressions. The first term in each expression corresponds to the clamped phase, and the second one to the unclamped, or sleep phase. In general Boltzmann Machines, this estimation is performed using stochastic methods. However, here the clamped term is easy to calculate, it requires summing a logistic type function over all training examples.

The same term is obtained by making the mean field approximation for the clamped phase in the general algorithm [Peterson and Anderson, 1987], which is exact in this case. It is more difficult to compute the sleep phase term, as it is an explicit sum over the entire input space, and within each term of this sum there is an implicit sum over the entire space of states of hidden units in the factor $Pr(\vec{x}|\phi_B)$. However, again taking advantage of the special structure of the combination model, we can reduce this sleep phase gradient term to a sum only over the states of the hidden units. Recall Equation (4.2):

$$Z_B = 2^n \sum_{\vec{h}\in\{0,1\}^m} \left[\exp(\sum_{i=1}^{m} h_i \theta^{(i)}) \prod_{j=1}^{n} \cosh(\vec{\omega}(\vec{h})_j) \right] .$$

A similar derivation gives that

$$Pr(\vec{h}|\phi_B) = \frac{\exp(\sum_{i=1}^{m} h_i \theta^{(i)}) \prod_{j=1}^{n} \cosh(\vec{\omega}(\vec{h})_j)}{\sum_{\vec{h'}\in\{0,1\}^m} \left[\exp(\sum_{i=1}^{m} h'_i \theta^{(i)}) \prod_{j=1}^{n} \cosh(\vec{w}(\vec{h'})_j) \right]} \tag{4.13}$$

The second term of the derivative w.r.t. $\theta^{(i)}$ is $\partial/\partial\theta^{(i)} \ln Z_B = (\partial/\partial\theta^{(i)} Z_B)/Z_B$. As $\theta^{(i)}$ appears only once in Z_B, we get that:

$$\frac{\partial}{\partial\theta^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x}\in S} \frac{1}{1+e^{-(\vec{\omega}^{(i)}\cdot\vec{x}+\theta^{(i)})}} - N \sum_{\vec{h}\in\{0,1\}^m} \Pr(\vec{h}|\phi_B) h_i \ . \tag{4.14}$$

Similarly, for each component of $\vec{w}i$, we use the fact that $d\cosh(t)/dt = \tanh(t)$, to get that

$$\frac{\partial}{\partial\omega_j^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x}\in S} x_j \frac{1}{1+e^{-(\vec{\omega}^{(i)}\cdot\vec{x}+\theta^{(i)})}} - N \sum_{\vec{h}\in\{0,1\}^m} Pr(\vec{h}|\phi_B) h_i \ \tanh(\vec{\omega}(\vec{h})_j) \tag{4.15}$$

The formulas for the gradients of the log likelihood for the real-valued model are very similar. A derivation similar to the one used to derive Equation (4.13), gives us that, for the real-valued model

$$\Pr(\vec{h}|\phi_R) = \frac{\exp(\sum_{i=1}^{m} h_i \theta^{(i)} + \frac{1}{2}||\vec{\omega}(\vec{h})||_2^2)}{\sum_{\vec{h'}\in\{0,1\}^m} \left[\exp(\sum_{i=1}^{m} h'_i \theta^{(i)} + \frac{1}{2}||\vec{\omega}(\vec{h})||_2^2) \right]} \ . \tag{4.16}$$

Using this equation we get that

$$\frac{\partial}{\partial\theta^{(i)}} \text{log-likelihood}(\phi_R) = \sum_{\vec{x}\in S} \frac{1}{1+e^{-(\vec{\omega}^{(i)}\cdot\vec{x}+\theta^{(i)})}} - N \sum_{\vec{h}\in\{0,1\}^m} Pr(\vec{h}|\phi_R) h_i \tag{4.17}$$

and for each j

$$\frac{\partial}{\partial\omega_j^{(i)}} \text{log-likelihood}(\phi_R) = \sum_{\vec{x}\in S} x_j \frac{1}{1+e^{-(\vec{\omega}^{(i)}\cdot\vec{x}+\theta^{(i)})}} - N \sum_{\vec{h}\in\{0,1\}^m} Pr(\vec{h}|\phi_R) h_i \vec{\omega}(\vec{h})_j \tag{4.18}$$

Equations (4.14-4.13) are very similar to Equations (4.17-4.16). The differences are in the partial derivative of the normalization factors, Z_B and Z_R, with respect to the weight vectors. Note that the equations for the real-valued model are simpler. As was discussed in Section (4.2.6), the normalization factors for the real and binary models are very close to each other when the weight vectors $\vec{w}i$ have small l_2 norm. Thus although the equations for the maximal likelihood solutions differ, the solution of the real-valued model are approximate solutions for the binary model and vice versa.

The time required to compute Equations (4.14) and (4.15), (or Equations (4.17) and (4.18) is $O(|S|n + 2^m)$. Thus, if m is small compared with the size of the sample S, then the computation time is linear in the number of training example and in the size of the input vector, which is reasonable. However, for large m it might not be possible to compute all 2^m terms. There is a way to avoid this exponential explosion if we can assume that a small number of terms dominate the sums. If, for instance, we assume that the probability that more than k hidden units are active (+1) at the same time is negligibly small we can get a good approximation by computing only $O(m^k)$ terms. In the extreme case where we assume that only one hidden unit is active at a time (i.e. $k = 1$), the combination model essentially reduces to the standard mixture model as discussed is Section 4.2.3. For larger k, this type of assumption provides a middle ground between the generality of the combination model and the simplicity of the mixture model. In the next section we show how the gradient of the real-valued model can be approximated when m is large.

4.3.2 Approximating the gradient

One possible approach to estimating the gradient when m is large is to search for the larger terms in Equations (4.17,4.18) and ignore the smaller ones. We now show that in the case of the real-valued model the problem of locating the large terms is equivalent to a simple geometric problem. Although this problem is NP-hard in the general case it might typically be easy in the cases that we encounter in real life problems.

Recall Equation (4.16)

$$Pr(\vec{h}|\phi_R) = \frac{\exp(\sum_{i=1}^m h_i\theta^{(i)} + \frac{1}{2}||\sum_{i=1}^m h_i\vec{w}i||_2^2)}{\sum_{\vec{h'}\in\{0,1\}^m}\left[\exp(\sum_{i=1}^m h'_i\theta^{(i)} + \frac{1}{2}||\sum_{i=1}^m h'_i\vec{w}i||_2^2)\right]}$$

We would like to estimate which of the vectors $\vec{h}$ correspond large terms. i.e. we would like to find all $\vec{h}$ such that $g(\vec{h}) = \sum_{i=1}^m h_i\theta^{(i)} + \frac{1}{2}||\sum_{i=1}^m h_i\vec{w}i||_2^2$ is large. Define the following matrix notation. We use $\vec{x}$ to denote a column vector and $\vec{x}^T$ to denote its transpose, i.e. a row vector. We define

$$\Omega = \begin{pmatrix} (\vec{w}1)^T \\ (\vec{w}2)^T \\ \vdots \\ (\vec{w}m)^T \end{pmatrix}$$

$$\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_m)^T$$

Using this notation we define $g(\vec{h})$ as

$$g(\vec{h}) = \vec{h}\cdot\vec{\theta} + \frac{1}{2}||\vec{h}^T\Omega||_2^2 \ ,$$

and rewrite Equation (4.16) as

$$Pr(\vec{h}|\phi_R) = \frac{\exp(g(\vec{h}))}{\sum_{\vec{h'}\in\{0,1\}^m}\exp(g(\vec{h'}))} \ .$$

Rearranging $g(\vec{h})$ we get

$$g(\vec{h}) = \frac{1}{2}||\vec{h}^T\Omega + \vec{\theta}^T(\Omega\Omega^T)^{-1}\Omega||_2^2 - \frac{1}{2}\vec{\theta}^T(\Omega\Omega^T)^{-1}\vec{\theta}\ ,$$

assuming that $\Omega\Omega^T$ is not singular.

The second term is a constant and is eliminated by the normalization. We can therefore ignore it. The first term corresponds to the distance between a sum of a subset of the weight vectors and the fixed vector $\vec{\theta}^T(\Omega\Omega^T)^{-1}\Omega$. The problem of finding the settings of $\vec{h}$ for which $g(\vec{h})$ is largest translates to the problem of finding a subset of a given set of vectors which is furthest away (in the regular Euclidean distance) from a given fixed vector.

It is not clear how hard this computation problem is in the general case. If the vectors are orthogonal then the problem is easy. In this case the set of all 2^m vector combinations defines the corners of a rectangular box. If the dot product, $\vec{x} \cdot \vec{w}i$, is equal to $||\vec{w}i||_2/2$ for all i, then $\vec{x}$ is in the center of the box. Any deviation from equality for a particular index i determines whether the vector corresponding vector, $\vec{w}i$, is in the subset whose sum is furthest from $\vec{x}$. In general, one of the closest subset-sums is equal to $\vec{h}^T\Omega$, where each coordinate of $\vec{h}$ is defined by:

$$h_i = \begin{cases} 1 & \text{if } \vec{w}i \cdot \vec{x} \leq \frac{1}{2}||\vec{w}i||_2 \\ 0 & \text{otherwise} \end{cases}$$

A promising direction for further research is to find methods that can solve this problem efficiently in the general case. Such methods would compute an approximation to the gradient by computing only the largest terms in the sum that defines it.

4.3.3 Projection Pursuit methods

A statistical method that has a close relationship with the combination model is the Projection Pursuit (PP) technique [Huber, 1985, Friedman *et al.*, 1984, Friedman, 1987]. In this section we give a short overview of the technique, show how it relates to the combination model, and present a learning algorithm for the combination model based on Projection Pursuit methods. This algorithm is a greedy algorithm that generates the hidden units one by one. It avoids the exponential blowup of the standard gradient ascent technique, and also has that advantage that the number m of hidden units is estimated from the sample as well, rather than being specified in advance.

4.3.4 Overview of Projection Pursuit

Many methods for analyzing high dimensional data study the first and second order statistics of the data, which are the mean vector and the covariance matrix. Principal components analysis is an example of such a method. Such methods necessarily ignore the structure of the distribution that is not reflected in the first and second order statistics, which may be an important part. Projection Pursuit methods can sometimes find this important high-order structure.

The distribution model over R^n with the largest entropy for a given average and covariance is a Gaussian distribution. Thus one natural definition of the information ignored by the second order analysis is the deviation of the empirical distribution from the Gaussian distribution. Low order linear projections have been traditionally used by researchers in their efforts to understand high dimensional distributions. As all projections of a Gaussian distribution produce a Normal marginal distribution. Thus, if a projection of a distribution generates a marginal distribution that is very different from the normal distribution, this is an indication that the projection contains information about the distribution that does not exist in its covariance matrix. Such a projection may be called an "interesting" projection. There are various "projection indices" defined in the PP literature to measure how interesting a particular projection is, and many of these indices relate directly to the deviation of the marginal distribution from a Normal distribution. Projection Pursuit methods locate the low dimensional projections in which the projection index is largest, i.e. those projections that are most interesting.

Originally, PP was used to suggest projection directions as an aid for the manual exploration of high dimensional data via two or three dimensional projections. Later PP became a complete method for statistical data analysis, using repeated search for interesting projections to generate n-dimensional density estimations. The search for a description of the distribution of a sample in terms of its projections can be formalized in the context of maximal likelihood density estimation in the following way [Friedman, 1987]. Define $p_0(\vec{x})$ to be the initial estimate of the density over R^n, i.e. the Gaussian density with appropriate mean and covariance. Define G to be a family of functions from R to R and A to be the set of vectors of length 1, i.e. $A = \{\vec{\alpha} \in R^n | \; ||\vec{\alpha}||_2 = 1\}$. Using these we define the nth order projection estimates to be the following set of densities

$$\mathcal{PP}_m = \left\{ \frac{1}{Z} p_0(\vec{x}) \prod_{i=1}^{m} g_i(\vec{\alpha}^{(i)} \cdot \vec{x}) \;\; \middle| \;\; \vec{\alpha}^{(i)} \in A; \;\; g_i \in G; \;\; Z = \int_{R^n} p_0(\vec{x}) \prod_{i=1}^{m} g_i(\vec{\alpha}^{(i)} \cdot \vec{x}) d\vec{x} \right\} \tag{4.19}$$

The log-likelihood of a specific density $p \in \mathcal{PP}_m$ with respect to a sample $S = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(N)}\}$, where $\vec{x}^{(i)} \in R^n$ is defined, in the standard way, to be

$$LL(p|S) = \sum_{\vec{x} \in S} \ln p(\vec{x}) \; .$$

The goal of Projection Pursuit is to find a series of approximations: $p_1 \in \mathcal{PP}_1$, $p_2 \in \mathcal{PP}_2, \ldots p_m \in \mathcal{PP}_m$ that have maximally increasing log-likelihood. The first approximation, p_0, is the Gaussian density itself, and the $(i+1)$-st approximation is generated by adding a factor $g_i(\vec{\alpha}^{(i)}) \cdot \vec{x})$ to the ith approximation. The vector $\vec{\alpha}^{(i)}$ is called the ith projection of the data.

The projection index is a function of $\vec{\alpha}^{(i)}$ that is a heuristic measure of the anticipated contribution of a factor involving the projection $\vec{\alpha}^{(i)}$ to the likelihood of the model. Given a choice of $\vec{\alpha}^{(i)}$, the *optimal* choice of the function $g_i(\cdot)$ in terms of maximizing the likelihood is the following [Friedman *et al.*, 1984]. Define $p_i^{\vec{\alpha}^{(i)}}(t)$ to be the marginal density on R generated by projecting the density p_i on the direction $\vec{\alpha}^{(i)}$. Similarly define $\hat{p}^{\vec{\alpha}^{(i)}}(t)$ to be an approximation to the marginal density generated by projecting the true density on the direction $\vec{\alpha}^{(i)}$, estimated empirically using the sample S.[7] Then the optimal choice for $g_i(\cdot)$, in terms of maximizing the likelihood of the model, is

[7] Note that the marginal density is a one dimensional function, thus the number of samples needed for estimating it

$$g_i(t) = \frac{\hat{p}^{\vec{\alpha}^{(i)}}(t)}{p_i^{\vec{\alpha}^{(i-1)}}(t)} \tag{4.20}$$

As the optimal choice of $g_i(\cdot)$, for a given choice of $\vec{\alpha}^{(i)}$ is simple to calculate. The main problem of designing a projection pursuit method is finding a good projection index whose calculation can be performed efficiently. Various projection indices have been discussed in the literature [Huber, 1985, Friedman, 1987]. Selection of a direction that has a high projection index is usually performed using gradient following methods. After a local maximum of the projection index has been found, the index function is altered to prevent the search from finding the same direction again, and a search for a direction with a high projection index is started from a different starting point.

The search for new projection directions can be simplified if instead of altering the projection index function, the sample is altered in a way that previously found interesting projections $(\vec{\alpha}^{(1)}, \vec{\alpha}^{(2)}, \ldots, \vec{\alpha}^{(i-1)})$ are made to appear uninteresting, i.e. Normally distributed. So called "structure removal" methods have been devised towards this goal [Huber, 1985, Friedman, 1987]. These methods alter the sample in such a way that a specific single projection that has been interesting is made uninteresting while all orthogonal projections are left unchanged. Put in another way, suppose that some density $p \in PP_m$ has high likelihood with respect to a given sample, and that one of the factors in p is $g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$. Then removing the structure corresponding to $g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$ means transforming the sample into a sample for which $p(\vec{x})/g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$, which is a model in $\mathcal{PP}_{m-1}$, has high likelihood.

To summarize, most iterative projection pursuit methods share the following common structure:

- **Initialization**

  Set S_0 to be the input sample.

  Set p_0 to be the initial density (Gaussian).

- **Iteration**

  Repeat the following steps for $i = 1, 2 \ldots$ until all projections of S_i are almost Normal.

  1. Find a direction $\vec{\alpha}^{(i)}$ for which the projection index of the projection of S_{i-1} is maximized.
  2. Approximate the actual marginal density in the direction $\vec{\alpha}^{(i)}$ by finding a close fit to the density of the projection of the sample S_{i-1}. Set $g_i(\cdot)$ to be the ratio between this approximation and the marginal density produced on $\vec{\alpha}^{(i)}$ by p_{i-1}, using Equation (4.20).
  3. Set S_i to be S_{i-1} with the structure defined by the factor $g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$ removed. This makes the projection of S_i on $\vec{\alpha}^{(i)}$ uninteresting, and all of the orthogonal projections remain equal to that of S_{i-1}.
  4. Set $p_i(\vec{x})$ to be $p_{i-1}(\vec{x}) g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$.

Notice that in this method the functions g_i are chosen in such a way that the product $\prod_{i=1}^{m} g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$ is normalized for each m and there is no need for an additional normalization term Z, as appears in the definition of $\mathcal{PP}_m$ in Equation (4.19).

---

is relatively small. In this way projection pursuit avoids, to some degree, the infamous "curse of dimensionality" in the estimation of the distribution of high dimensional data.

Projection Pursuit has proved itself successful in some experiments [Friedman, 1987]. However, the search for best density is performed in a greedy manner and might not succeed in finding the optimal density in $\mathcal{PP}_m$. While there is quite a large body of research on the representational power of projection pursuit models, little is theoretically known about reliability of the associated learning algorithms, such as the one presented above.

4.3.5 Projection Pursuit and the combination model

Recall Equation (4.6), which describes the density generated by the real-valued combination model:

$$p(\vec{x}) = e^{-\frac{1}{2}||\vec{x}||_2^2} \frac{1}{Z_R} \prod_{i=1}^{m} \left(1 + e^{\theta_i + \vec{w}i \cdot \vec{x}}\right).$$

Using the following definitions we see that this class of models is a special case of the class of models presented in Equation (4.19).

$$p_0(\vec{x}) = (2\pi)^{-n/2} e^{-\frac{1}{2}||\vec{x}||_2^2} = \mathcal{N}(0,1)$$

$$\vec{\alpha}^{(i)} = \frac{\vec{w}i}{||\vec{w}i||_2}$$

$$G = \left\{ g : R \to R \;\mid\; g(t) = \frac{1}{Z}\left(1 + e^{\theta_i + t||\vec{w}i||_2}\right) \;\; ; Z \in R \right\}$$

It is clear that, under these definitions, $p(\vec{x})$ is a function in $\mathcal{PP}_m$. In the next section we present a greedy algorithm for learning the combination model that is based on this relation.

A similar relationship holds for the binary model. However, we have not managed to find a good structure removal procedure for the binary-valued model. We thus present an algorithm for learning the real-valued model and, based on the relations given in Section 4.2.6, we claim that the solutions that we find for the real-valued model are approximate solutions for the binary-valued model.

There are two main differences between our work and previous work on using exploratory projection pursuit algorithms for estimating distributions. The first difference is that while our model defines a distribution on all R^n, our data-points are taken from $\{-1,+1\}^n$. However, as discussed at the end of Section 4.2.6, the projections of the binary vectors generate marginal distributions that are close to Normal, similarly to the distributions we expect from real-valued data.

The second difference is that the family of functions G from which the g_is are taken is a very restricted set of functions. This is unlike standard PP techniques, in which the functions g_i are chosen from some very broad family, such as some family of spline functions. This means that, in our case, any single function $g \in G$ might be far from adequate for describing the marginal distribution on some direction $\vec{\alpha}^{(i)}$ and several factors with the same $\vec{\alpha}$ might be needed. This, in turn, has the effect that eliminating the structure generated by a single factor does not amount to transforming the marginal distribution on the corresponding projection so that it becomes completely uninteresting. As most structure elimination techniques do exactly that, they are unfit in the context of learning the combination model.

4.3.6 PP algorithm for learning the combination model

In this section we present a variant of PP that is a learning algorithm for the combination model. Our algorithm combines the search for an interesting projection direction, $\vec{\alpha}$, with the search for the corresponding projection function, $g(\cdot)$. The algorithm searches for the optimal factor by maximizing the likelihood of a single factor model with respect to the (possibly altered) sample. After such a factor is found, the algorithm alters the examples in such a way that the structure encoded in the factor is eliminated, and subsequent searches will find different factors.

The algorithm is thus based on two elements. The first element is a method for finding a maximal likelihood combination model with a single hidden unit. This method serves both for finding a projection direction, and for finding the function $g_i(\cdot)$ associated with this direction. The second element is a structure removal procedure. We shall describe the two elements in turn.

We have previously described how gradient ascent can be used for finding model with highest log-likelihood. However, for the special case where there is only a single hidden unit in the model, a much faster method can be used. This method is an Expectation-Maximization (EM) method [Dempster *et al.*, 1977]. EM is a general method for estimating the parameters of distribution models that have both observable and unobservable random variables. This method achieves extremely fast convergence when used for estimating a mixture of product distributions.[8]

The Expectation Maximization method is based on iterative improvement of the estimates of the maximal likelihood values of the model parameters. It starts with some initial guess of the parameters ϕ_{init}, and proceeds by iterating the following two steps. It can be shown [Dempster *et al.*, 1977], that each of these iterations improves the likelihood of the parameters.

1. Using the old setting of the parameters, ϕ_{old}, as if they were the actual parameters, some statistics of the joint distribution of the hidden and the observable variables are calculated.
2. The old setting of the parameters, ϕ_{old}, is replaced with a new setting of the parameters ϕ_{new}, which is the most likely setting of the parameters given the values of the statistics calculated in step 1. These new parameters are used as the old parameters in the following iteration.

To see how this method is implemented for the problem of estimating the parameters of a real-valued combination model with a single hidden unit let us calculate the maximal likelihood setting of the parameters assuming that we are given a sample S', of size N, in which each element describes the value of both the observable random variables, $\vec{x}$, and the unobservable random variable h. The log likelihood is

$$LL(\theta,\vec{\omega}|S') = \sum_{(h,\vec{x})\in S'} \ln P(\vec{x},h|\theta,\vec{\omega}) = \sum_{(h,\vec{x})\in S'} \ln \frac{\exp\left(h(\theta+\vec{\omega}\cdot\vec{x})\right)}{Z_R}$$

$$= \sum_{(h,\vec{x})\in S'} h(\theta+\vec{\omega}\cdot\vec{x}) - N(2\pi)^{n/2} \ln\left(1+\exp\left(\theta+\frac{1}{2}||\vec{\omega}||_2^2\right)\right)$$

[8] It is not easy to implement EM directly on the complete combination model, because although this distribution can be expressed as a mixture of product distributions, the parameters that define the mixture components are coupled.

Taking the derivative of the log-likelihood with respect to the parameters and equating to zero to find the optimal setting of the parameters, we get the following equations. From the derivative w.r.t. θ we get that

$$\sum_{(h,\vec{x})\in S} h = N \text{ logistic}\left(\theta_{\text{opt}} + \frac{1}{2}||\vec{\omega}_{\text{opt}}||_2^2\right) , \tag{4.21}$$

and from the gradient w.r.t. $\vec{\omega}$ we get that

$$\sum_{(h,\vec{x})\in S} h\vec{x} = \vec{\omega}_{\text{opt}} N \text{ logistic}\left(\theta_{\text{opt}} + \frac{1}{2}||\vec{\omega}_{\text{opt}}||_2^2\right) \tag{4.22}$$

Notice that if we divide the sums on the left hand side of Equations (4.21) and (4.22) by N, we get the definition of the empirical estimates of $E(h)$ and of $E(h\vec{x})$, which we shall denote by $\hat{E}(h)$ and $\hat{E}(h\vec{x})$. Solving Equations (4.21) and (4.22) for the values of the optimal parameters, we get that:

$$\vec{\omega}_{\text{opt}} = \frac{\hat{E}(h\vec{x})}{\hat{E}(h)} , \tag{4.23}$$

and

$$\theta_{\text{opt}} = -\ln\frac{1-\hat{E}(h)}{\hat{E}(h)} - \frac{1}{2}||\vec{\omega}_{\text{opt}}||_2^2 . \tag{4.24}$$

We thus see that the statistics that we need to estimate in the first step of the EM iteration are $\hat{E}(h)$ and $\hat{E}(h)$. These statistics can be directly calculated from the sample S', as this sample includes both $\vec{x}$ and h. However, given a setting of the parameters, we can compute the distribution of h for any setting of $\vec{x}$, and thus calculate the desired statistics.

The implementation of the EM method for the combination model with a single hidden unit is thus as follows. We start with an initial setting of the parameters: $(\vec{\omega}_{\text{init}}, \theta_{\text{init}})$ and proceeds by iterating the following two steps on the given sample $S = \langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N \rangle$

1. In the Expectation calculation step the current parameters $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ are used as if they describe the correct input distribution. Given this description and a particular setting of the input units, $\vec{x}$, we can compute probability that each hidden unit is 0 or 1 given any setting of the observable vector $\vec{x}$:

   $$\Pr(h_i = 1|\vec{x}, \vec{\omega}_{\text{old}}, \theta_{\text{old}}) = \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}})\vec{x} .$$

   Using this equation and the sample S, it is possible to compute the following estimates:

   $$\hat{E}(h\vec{x}) = \frac{1}{N}\sum_{\vec{x}\in S} \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}})\vec{x}$$

   $$\hat{E}(h = 1) = \frac{1}{N}\sum_{\vec{x}\in S} \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}})$$

2. In the Maximization step, new parameters $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ are calculated using Equations (4.23) and (4.24). The new parameters $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ are used as the old parameters $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ in the following iteration.

3. The iteration terminates when the difference between $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ and $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ becomes insignificant.

We now present the structure removal procedure. In the analysis of the real-valued model in Section (4.2.3) we have shown that the addition of a hidden variable has the effect of replacing the previous distribution by a mixture of two distributions, the first of which is equivalent to the previous, and the second is a shifted copy of the previous distribution, shifted by the weight vector $\vec{w}i$ that corresponds to the hidden unit. The shifted copy corresponds to the case in which $h_i = 1$ while the unshifted one correspond to the case where $h_i = 0$. For each data point we compute the probability, p, that $h_i = 1$. We then flip a random coin whose bias is p and, according to the outcome of the coin flip, either keep the example as it is or subtract $\vec{w}i$ from it. This has the effect of shifting the shifted copy, which corresponds to $h_i = 1$ to coincide with the unshifted copy, which corresponds to $h_i = 0$. In this way the structure encoded by the hidden unit is eliminated from the empirical distribution. Details are described below.

- **Initialization**

  Set S_0 to be the input sample.

  Set p_0 to be the initial distribution (Gaussian).

- **Iteration**

  Repeat the following steps for $i = 1, 2 \ldots$ until no single-variable combination model has a significantly higher likelihood than the Gaussian distribution with respect to S_i.

  1. Perform an EM procedure to maximize the log-likelihood of a single hidden variable model on the sample S_{i-1}. Denote by θ_i and $\vec{w}i$ the parameters found by this procedure, and create a new hidden unit with associated binary random variable h_i with these weights and bias.
  2. Transform S_{i-1} into S_i using the following structure removal procedure.
     For each example $\vec{x} \in S_{i-1}$ compute the probability that the hidden variable h_i found in the last step is 1 on this input:

     $$P(h_i = 1) = \left(1 + e^{-(\theta_i + \vec{w}i \cdot \vec{x})}\right)^{-1}$$

     Flip a coin that has probability of "head" equal to $P(h_i = 1)$. If the coin turns out "head" then add $\vec{x} - \vec{w}i$ to S_i else add $\vec{x}$ to S_i.
  3. Set $p_i(\vec{x})$ to be $p_{i-1}(\vec{x}) Z_i^{-1} \left(1 + e^{\theta_i + \vec{w}i \cdot \vec{x}}\right)$, where $Z_i = \sum_{\vec{x}} p_{i-1}(\vec{x}) \left(1 + e^{\theta_i + \vec{w}i \cdot \vec{x}}\right)$.

4.4 Experimental work

We have carried out several experiments to test the performance of unsupervised learning using the combination model. The goals of these experiments is to show that the combination model is a useful one and to compare the performance of the different learning algorithm that we have developed.

The first set of experiments compares the two learning methods for the combination model presented in this paper. The first is the gradient ascent method, and the second is the projection pursuit method. The experiments in this set were performed on synthetically generated data. The input consisted of 4,000 binary vectors of 64 bits that represent 8×8 binary images. The binary vectors are synthesized using

a combination model with 10 hidden units whose weights were set as in Figure (4.3,a). Each square in this image denotes a single real valued parameter,[9] the matrix corresponds to the weight vector, and the rectangle above the matrix corresponds to the bias parameter θ. We shall refer to each random binary vector as an instance.

The ultimate goal of the learning algorithms was to retrieve the model that generated the instances, which we call the "target" model. However, this goal is generally not achievable. The first reason is that the optimal model is not unique, i.e. there usually are other combination models that generate the exact same distribution as the target model, or a distribution that is very close to it. For example, a permutation of the hidden units does not change the distribution defined by the model. As we have found out in the experiments, other simple transformations of the target model produce models that are almost as good as the target model. Another reason that we cannot retrieve the exact target is that the parameter vector of the target is real valued, and thus cannot be exactly identified by a finite number of instances. The third reason is that our algorithms are not guaranteed to find the optimal model for the given data. The gradient ascent algorithm is only guaranteed to locate a local maximum of the likelihood, and the Projection Pursuit algorithm is only guaranteed to increase the likelihood of the model with each additional hidden unit.

While the difference between the parameter vectors of the learned model and of the target model is usually large, their performance as models of the random instances is similar. We measure this performance using three different error measures. Each error measure defines a way of computing the error of a combination model with respect to a set of instances. We have measured these errors for the target model and for each of the learned models. Each measurement was taken both with respect to the instances that were used for learning (the "training" instances) and with respect to an independent test set of 4000 instances.

We now describe each of the three measures of error that we have used:

- *Average log-loss*

  Each learned distribution model defines a probability distribution, P, on the space of images. A popular measure of the distance between P and the actual distribution Q is the cross entropy, which is defined as $-\sum_x (Q(x) \log P(x))$. The cross entropy is minimized when $P = Q$, and is then equal to the entropy of Q. The cross entropy can be estimated by taking the average value of minus log of the probability that the model assigns to each instance in the sample. This measure of error is also called the log-loss error. We scale the error so that the uniform distribution model, that assigns equal probability to all instances, has an expected error of 1. The log-loss error is hard to compute for large combination models, which is why we use it only in the experiments on synthetic data in which we use only 10 hidden units in the models.

- *Single bit completion*

  We estimate the average number of mistakes made by the model when it is used to predict the value of single bits of the instances. More precisely, the mistakes it makes when used to predict the value of each single bit in each of the instances in the sample, when given the values of all the other bits of

[9] The results are given using Hinton diagrams [Rumelhart and McClelland, 1986], i.e. positive values are displayed as full rectangles, negative values as empty rectangles, and the area of the rectangle is proportional to the absolute value.

that instance. The combination model defines a probability for any possible instance. The prediction is defined as the value of the bit that corresponds to the more probable instance. We estimate this average number by choosing at random 5 bit locations for each instance in the sample.

- *Input reconstruction*

  We estimate the quality of the combination model as an input representation scheme. For each instance $(x_1, \ldots, x_n)$ we compute the most probable state of the hidden units. This state can be seen as an encoding of the instance. One way of defining the quality of this encoding scheme is to measure how much additional information is required to reconstruct the instance from the state of the hidden units alone. Each state of the hidden units defined a Bernoulli product distribution over the images. The additional information that is required to encode a particular instance is the log of one over the probability assigned to the instance. As the distribution is a Bernoulli product, this can be written as the following sum:

$$H(\vec{x}|\vec{h}) = \frac{1}{2}\sum_{i=1}^{n}\left[(1+x_i)\log_2 p_i + (1-x_i)\log_2(1-p_i)\right] \ ,$$

  where p_i is the independent probability of the ith input bit to be $+1$ given the hidden state, which is equal to

$$p_i = \text{logistic}\left(\sum_{j=1}^{m}\omega_i^{(j)}h_j\right)$$

  This measure of error is scaled so that it measures the additional information that is required per input bit.

All experiments used a test set and a separate training set, each containing 4000 examples. The gradient ascent method is based on the binary distribution model. It typically needed about 1000 epochs to stabilize.[10] In the projection pursuit algorithm, 4 iterations of EM per hidden unit proved sufficient to find a stable solution. The results are summarized in the following table and in Figure (4.3).[11]

	log-loss		single bit prediction		input reconstruction	
	train	test	train	test	train	test
gradient ascent for 1000 epochs	0.399	0.425	0.098	0.100	0.311	0.338
projection pursuit	0.893	0.993	0.119	0.114	0.475	0.480
Projection pursuit followed by gradient ascent for 100 epochs	0.411	0.430	0.091	0.089	0.315	0.334
Projection pursuit followed by gradient ascent for 1000 epochs	0.377	0.405	0.071	0.082	0.261	0.287
true model	0.401	0.396	0.077	0.071	0.286	0.283

The best learning result was achieved by starting with the projection pursuit algorithm then using the parameter vector that was learned as a starting point for the gradient ascent algorithm. The final result

[10] The algorithm used a standard momentum term (see [Hertz *et al.*, 1991], page 123) to accelerate the convergence.

[11] The difference between the measurements of the quality of the true model on the test set and on the training set are due to the random fluctuations between the two sets of examples. These differences provide an indication of the accuracy of our measurements.

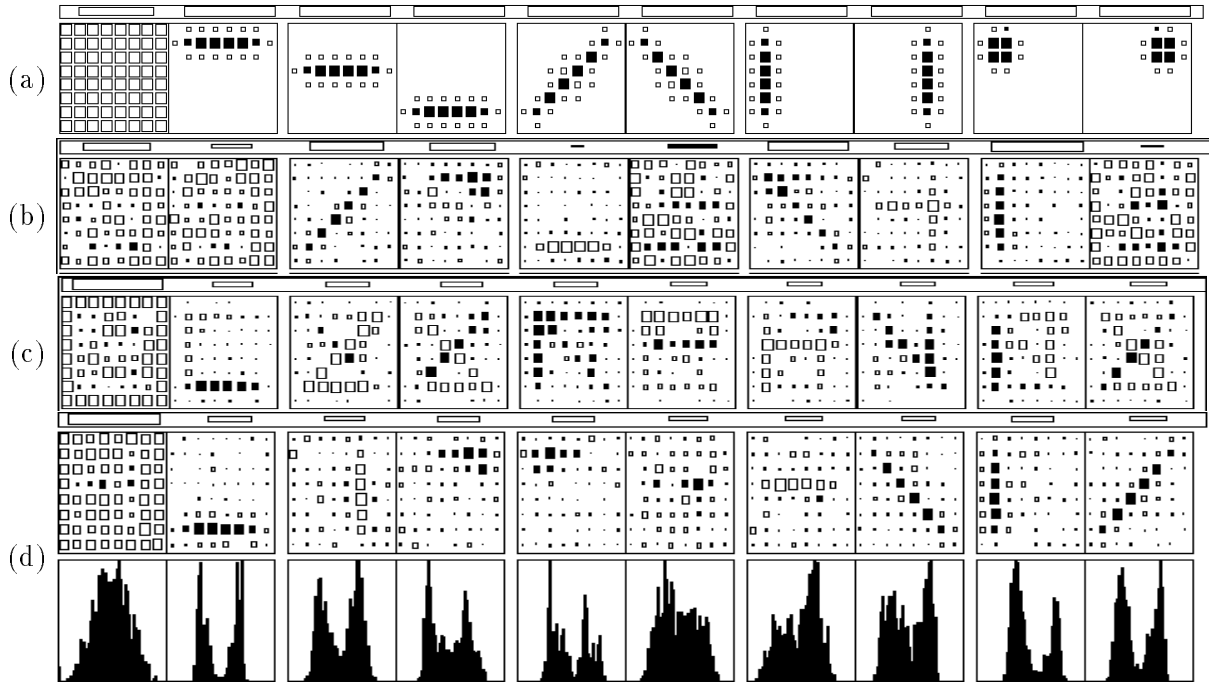


Figure 4.3: The weight vectors of the models in the synthetic data experiments. Each matrix represents the 64 weights of one hidden unit. The range of the weights is $[-6, +6]$ with the large white squares representing the value 6. The square above the matrix represents the units bias. positive weights are displayed as full squares and negative weights as empty squares, the area of the square is proportional to the absolute value of the weight. (a) The weights in the model used for generating the data. (b) The weights in the model found by gradient ascent alone. (c) The weights in the model found by projection pursuit alone. (d) The weights in the model found by projection pursuit followed by gradient ascent. For this last model we also show the histograms of the projection of the examples on the directions defined by those weight vectors; the bimodality expected from projection pursuit analysis is evident.

of this combination is presented in Figure 4.3(d), together with the corresponding projections of the data along the directions defined by the weight vectors. We can see that there is a close correspondence between the weight vectors in the learned model and the vectors in the target model described in Figure 4.3(a). Counting from left to right, the weight vectors of units 1,2,8,9, and 10 in the learned model are almost identical to the weight vectors of units 1,4,6,7,and 5 in the target model. Units 3 and 7 in the learned model are close to the negation of units 8 and 3 in the target model, and units 4 and 5 in the learned model are combinations of units (10,2) and (9,2) of the target model respectively. There is no exact correspondence of the biases. As we see from the table, the performance of the learned model is almost as good as that of the target model according to all three measures. We thus conclude that reversing the sign of weight vectors and combining them can sometimes create a different combination model whose corresponding distribution is very similar.

When the gradient ascent model is used to learn by itself (Figure 4.3(b)),it tends to get stuck in local

minima, as can be seen in the table. It is also a very slow method, both because of the large number of iterations that is required and because each iteration requires complex calculations. The fact that the local search process is stuck in a sub-optimal solution can be seen in the weight vectors of the learned model in that four of the weight vectors (those of units 1,2,6,10, counting from the left) have no clear correspondence to any of the weight vectors in the target model.

The Projection Pursuit method is very fast, but its results are weaker than those of the gradient ascent method by itself. It tends to find a model whose weight vectors correspond to various combinations of the weight vectors of the target model and their negations. The performance of the results of projection pursuit are similar to those of the gradient method in the single bit prediction measure and in the input reconstruction measure. On the other hand, the performance of the Projection pursuit model in terms of the likelihood of the model that it generates is very poor. The reason is that the data that we use is generated by a binary valued combination model, while the projection pursuit model is based on a real valued combination model. The difference between these two models is large, because the weights that are used in the target model are in the range $[-6, +6]$. As we have shown in Section 4.2.6, the binary model and the real valued model are approximately equal when the weights are small. To show that this is indeed the source of the error, we repeated the previous experiments using a target model with the weight vectors divided by a factor of 7, so that now all the weights are in the range $[-6/7, +6/7]$. The results are summarized in the following table

	log-loss		single bit prediction		input reconstruction	
	train	test	train	test	train	test
True Model	0.939	0.941	0.36	0.36	0.86	0.87
gradient ascent for 400 epochs	0.937	0.944	0.36	0.37	0.86	0.87
projection pursuit	0.964	0.966	0.38	0.39	0.92	0.92
Projection pursuit followed by gradient ascent for 400 epochs	0.935	0.943	0.36	0.37	0.86	0.87

We see that in this case, the likelihood of the model found by the projection pursuit algorithm is similar to that of the other models. Because in this case the weights are so small, the difference between the distribution defined by the model and the uniform distribution is small, as is reflected in the measures of accuracy. However, the difference from the uniform distribution is statistically significant. The combination of the two learning algorithms was able to retrieve the weights of the target model almost as well as in the previous experiment (see Figure 4.4).

In the second set of experiments we compare the performance of the combination model to that of the mixture model. The comparison uses real world data extracted from the NIST handwritten data base.[12] Examples are 16×16 binary images (see Figure (4.5)). There are 500 examples in the training set and 500 in the test set. We use 45 hidden units to model the distribution in both of the models. Because of the

[12]NIST Special Database 1, HWDB Rel1-1.1, May 1990.

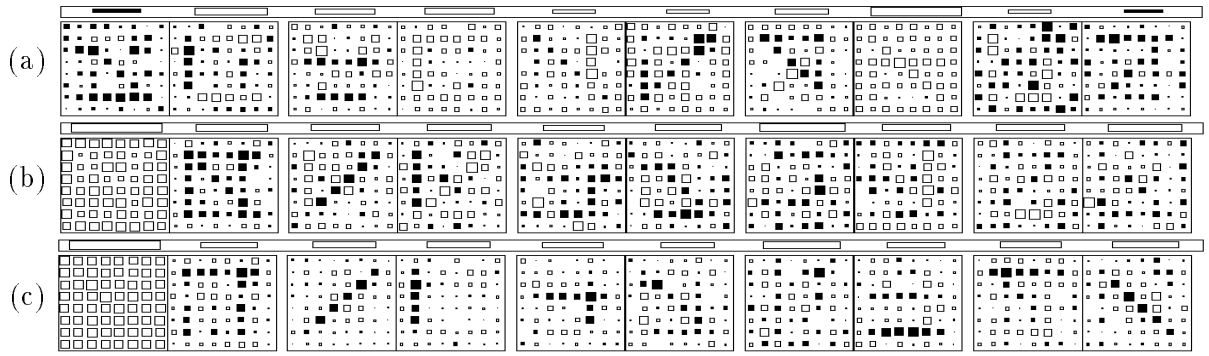


Figure 4.4: The weight vectors of the models in the synthetic data experiments. The target target is the same as in the previous experiment but the range of the weights is divided by a factor of 7, so that the largest white squares represent the value of 6/7. (a) The weights in the model found by gradient ascent alone. (b) The weights in the model found by projection pursuit alone. (c) The weights in the model found by projection pursuit followed by gradient ascent.

large number of hidden units we cannot use gradient ascent learning and instead use projection pursuit. For the same reason it was not possible to compute the likelihood of the combination model and only the other two measures of error were used. Each test was run several times to estimate the accuracy of our measurements.

For learning a mixture model we use an incremental version of EM. We start with a model with a single Bernoulli product distribution and run EM until the method converges. We then take a mixture of two Bernoulli product distributions, each of which is initialized to be a slight random perturbation of the single Bernoulli product. We then let EM run on this model until it converges, and then we split each component into two in a similar way. Continuing in this fashion we repeatedly double the size of the model.[13]

The final errors of many runs of these algorithms, starting from different initial weights, are summarized in the table below. The errors of two representative runs are given in Figures 4.8 and 4.9. A sample of the final weight vectors of the learned combination model and mixture model are given in Figures 4.6 and 4.7 respectively. A complete list of all of the 45 weight vectors for each model are given in Figures 4.10 and 4.11.

	single bit prediction		input reconstruction	
	train	test	train	test
Product distribution	0.29 ± 0.01	0.30 ± 0.01	0.78 ± 0.01	0.80 ± 0.01
Mixture model	0.19 ± 0.01	0.26 ± 0.01	0.55 ± 0.01	0.70 ± 0.01
combination model	0.19 ± 0.01	0.20 ± 0.01	0.60 ± 0.01	0.64 ± 0.01

The first line in this table, named "Product distribution" summarizes the performance of a simple distribution model that assumes that the pixels are distributed according to a Bernoulli product distribution. The reconstruction of the input, in this case, is simply the fixed reconstruction in which each bit is set to its more probable value. The performance of this model provides a baseline with respect to which we can

[13]When 32 units are to be split, only the first 13 of them are split, to give the final number of 45 mixture components.

compare the performance of the other distribution models whose goal is to capture dependencies between the pixels. We see that the performance of the combination model is significantly better than that of the mixture model on the test set. The difference is especially significant when compared to the baseline of the Product distribution model. Also, we see that the difference between the performance on the test set and on the training set, i.e. the over-fitting, is much smaller for the combination model.

A qualitative comparison between the weight vectors found by the two models confirms the expected advantage of the combination model in describing combinations of correlations. While the typical weight vectors of the mixture model (see Figure (4.7)), which is a sample out of Figure (4.10)) look very much like an average prototype of a specific digit, the weight vectors of the combination model relate to more local features, such as lines and curves (see Figure (4.6)), which is a sample out of Figure (4.11)). This relates to fact that the mixture model relates each example with the single weight vector that is most similar to it, while the combination model relates each example with a combination of its weights.

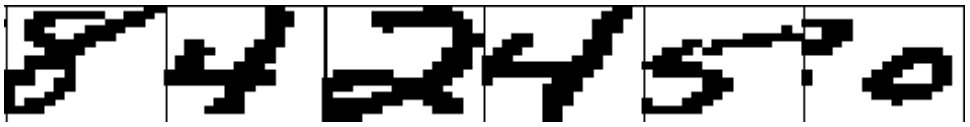


Figure 4.5: A few examples from the handwritten digits sample.

As the experiments on synthetic data have shown that PP does not reach optimal solutions by itself we expect the advantage of the combination model over the mixture model to increase further by using improved learning methods. Of course, the combination model is a very general distribution model and is not specifically tuned to the domain of handwritten digit images, thus it cannot be compared to models specifically developed to capture structures in this domain. However, the experimental results support our claim that the combination model is a simple and tractable mathematical model for describing distributions in which several correlation patterns combine to generate each instance.

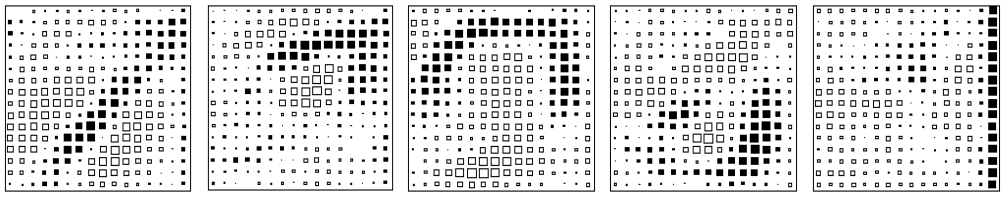


Figure 4.6: Typical weight vectors found by the combination model

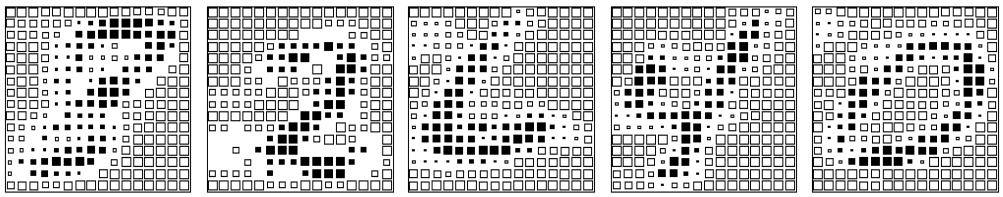


Figure 4.7: Typical weight vectors found by the mixture model

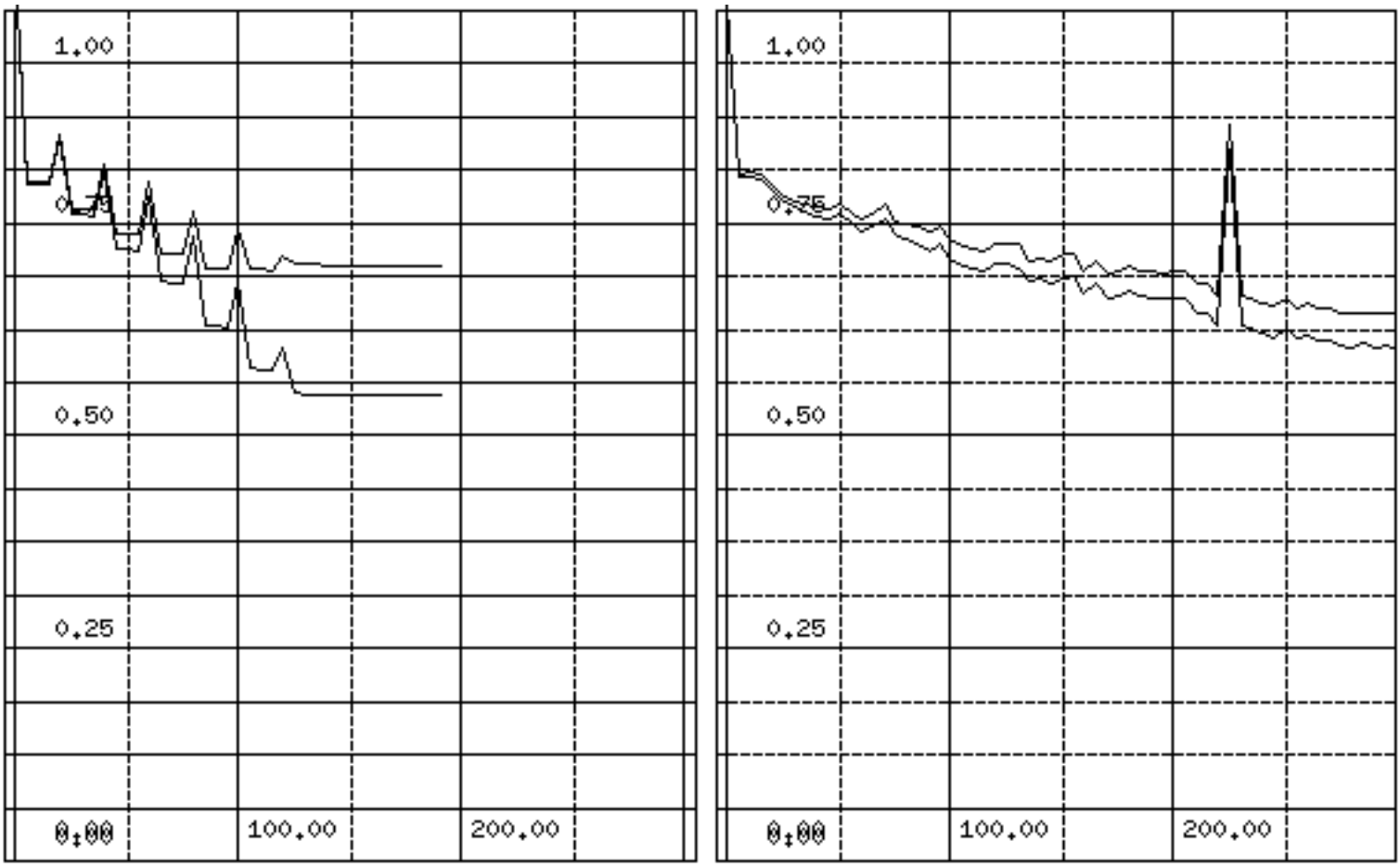


Figure 4.8: A comparison of the input reconstruction error on 16×16 pixel digit images. This error measures the average amount of additional information that is required for reconstructing the input from the state of the hidden units. The information is measured in bits per pixel. The higher and lower curves in each graph describe the error on the test set and on the training set respectively. The graph on the left describes the error of the mixture model as a function of the number of training iterations (epochs). The number of mixture components is doubled every 20 iterations. There is a spike in the error immediately following the doubling, as a result of the added randomization. The graph on the right describes the error of the combination model as a function of the number of iterations. (The spike in the graph around iteration 230 is a side effect of a "backfitting" stage that has not proven to be useful.)

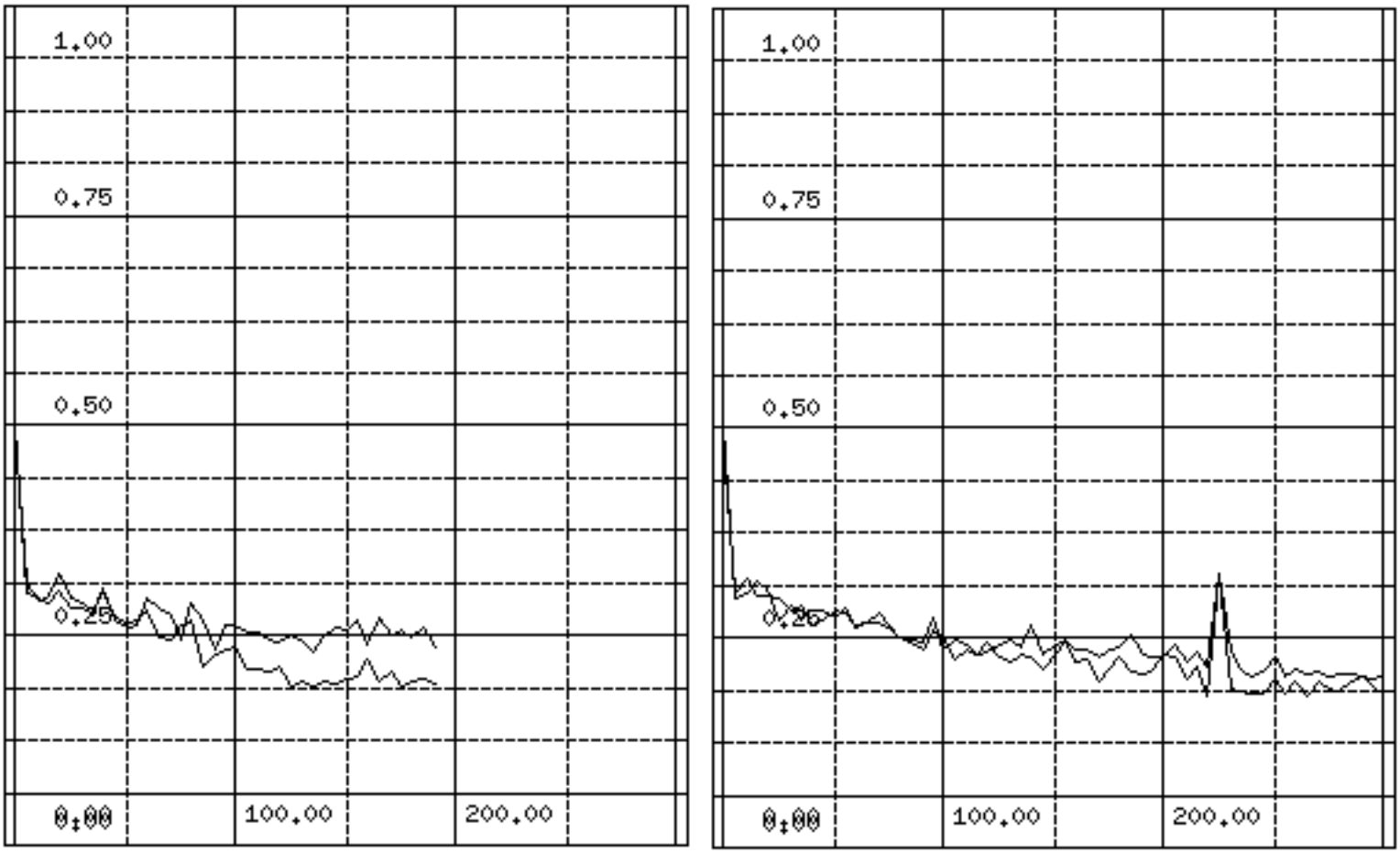


Figure 4.9: A comparison of the single bit completion error on 16×16 pixel digit images. The error measures the probability of a mistake in predicting a random single missing bit in the image, using the distribution model and the values of all the rest of the pixels. The higher and lower curves in each graph describe the error on the test set and on the training set respectively. The graph on the left describes the error of the mixture model as a function of the number of training iterations (epochs). The number of mixture components is doubled every 20 iterations. The graph on the right describes the error of the combination model as a function of the number of iterations. (The peak in the graph around iteration 230 is a side effect of a "backfitting" stage that has not proven to be useful.)

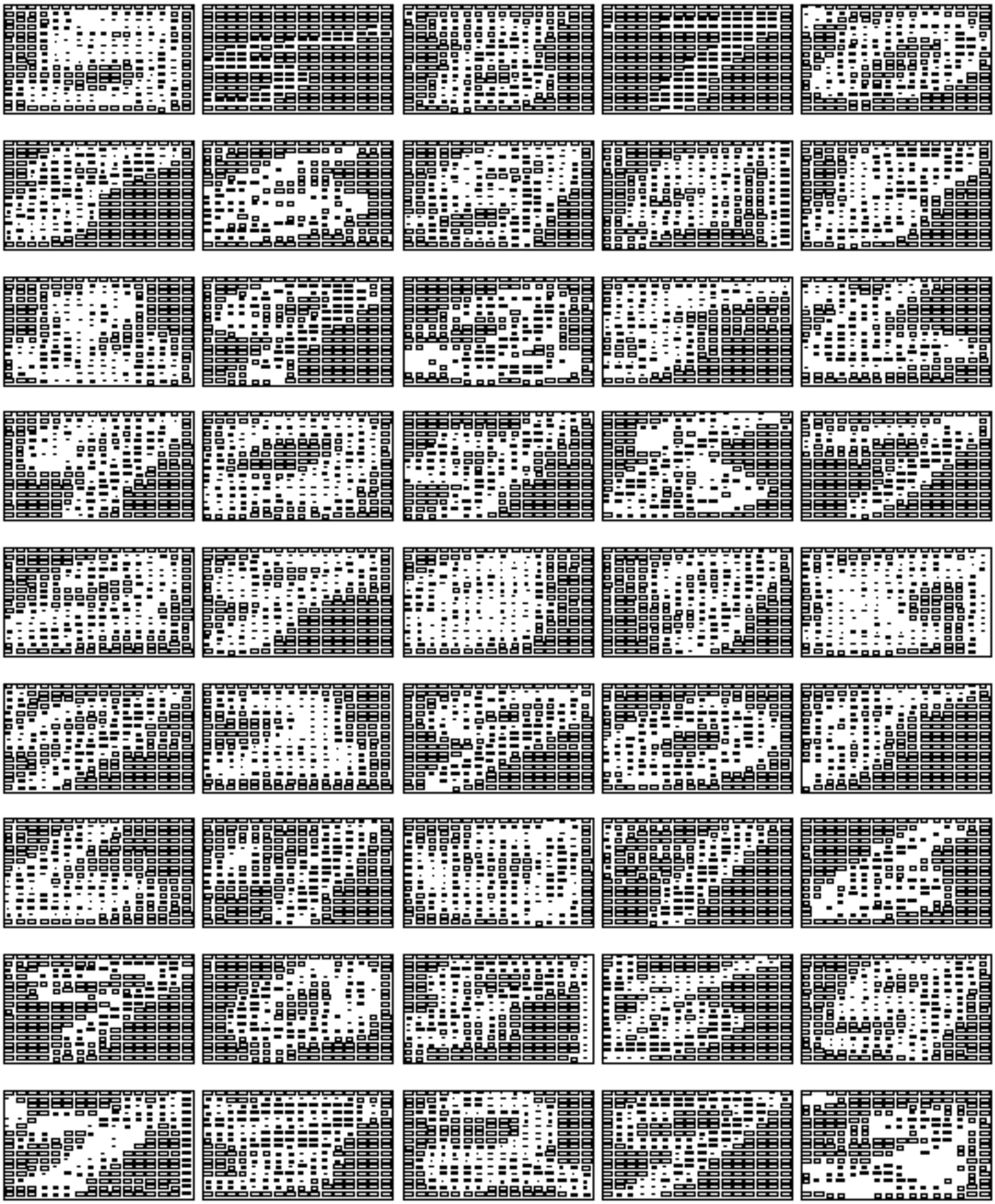
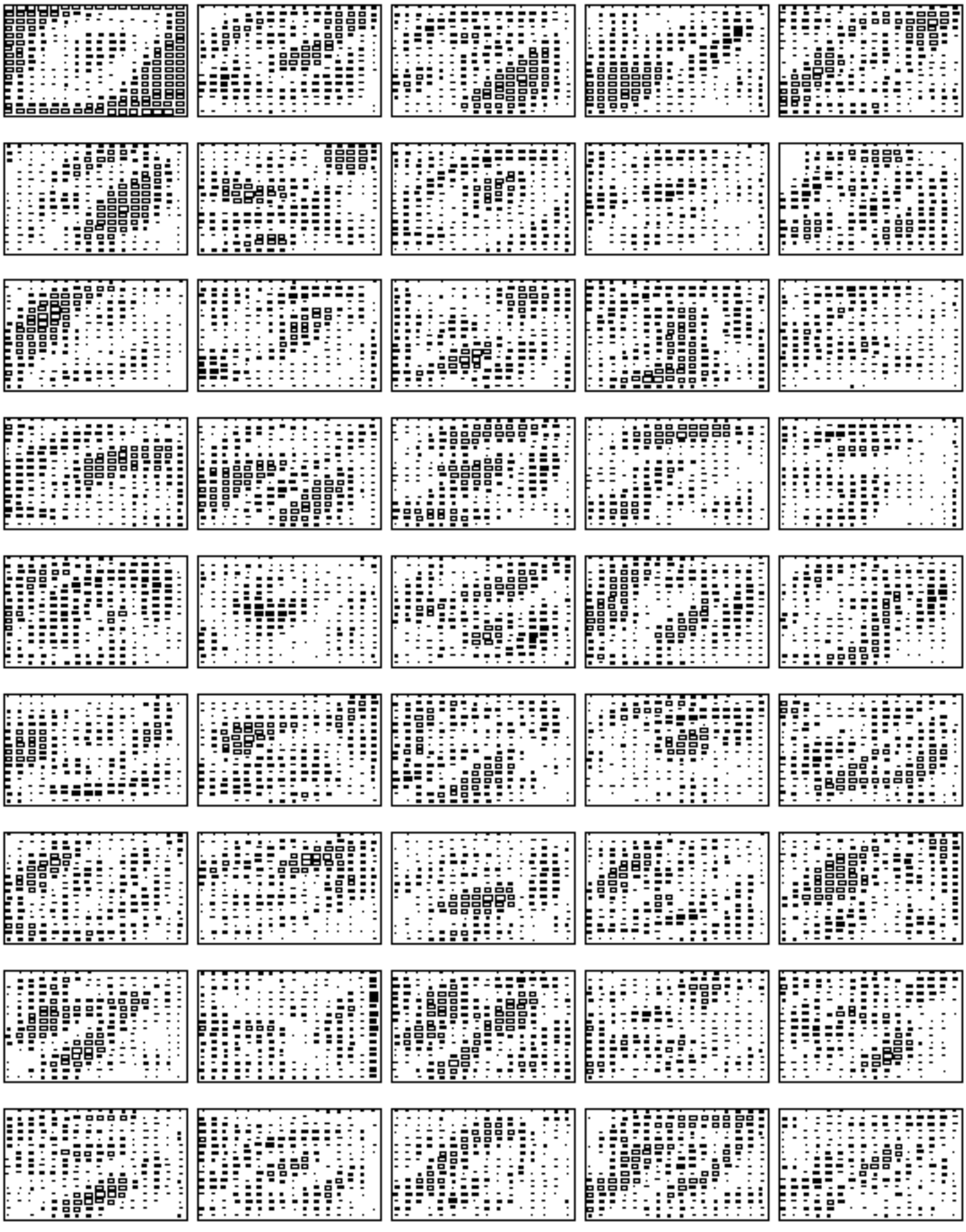


Figure 4.10: The weight vector, or image templates, found by the the mixture model



5. Concluding remarks

Each chapter of this thesis discusses different learning algorithms, aiming at different goals, and analyzed using a different mathematical framework. However, a common theme in much of this work is the important role that the distribution of instances plays in learning.

In this section we speculate on the possible application of our learning algorithms to real-world problems. We do that by discussing in some detail a simple real world classification problem. Through this example we illustrate some phenomena that re-occur in our work and point out some advantages our algorithms might have in real world problems, as well as some of the technical and conceptual problems that lie on the way to a more complete understanding of this type of learning algorithm.

Suppose that we wish to build a machine that will learn to sort apples of two varieties, for example, Pippin and Mutzu, according to their color. Suppose, for simplicity, that we measure the color using two color filters, which give us two real valued measurements for each apple. Thus each apple corresponds to a point in the plane, which is labeled either "P" or "M". The goal of learning is to find a mapping from the plane to the set $\{P, M\}$ that optimally predicts the type of an apple from measurements of the apples color. The problem setup is illustrated in Figure 5.1(a).

There might be apples whose color is not representative of their variety, these apples correspond to points whose label does not agree with that of the optimal prediction rule. However, we expect that if the two color filters are properly chosen then the label of most apples will agree with the prediction rule. Also, it is reasonable to expect that the instances whose label disagrees with the prediction rule are located close to the borders between the two labels. If more than two colors are used to make the predictions, then we can expect the optimal prediction rule to improve, and the concentration of the error close to the borders to become more pronounced.

One approach to the problem, that is suggested by our boosting algorithm, described in Chapter 2, is to create a large set of simple hypotheses and combine them using a majority vote. As we expect most of the typical examples to be far from the decision border and have equal labels, the following simple algorithm is likely to generate prediction rules that have a considerable advantage over random predictions:

1. Pick a small sample of labeled apples, i.e. labeled points in the plane.
2. For each point in the sample, find the largest disk around it that contains only points with the same label.
3. Choose the disk that contains the largest number of (equally labeled) points.
4. Output the hypothesis that labels the points in the disk with the label of the point in its center, for points outside the disk, the hypothesis generates a random label by flipping a fair coin.

The type of hypotheses that this algorithm is likely to generate is given in Figure 5.1(c). The result of running the boosting by majority algorithm using this weak learner is a randomized rule for predicting labels that combines a large number of such disks, as described in Figure 5.1(d). Given a specific point on the plane, each disk contributes one vote, if the point is inside the disk, then the vote is according to the label associated with the disk, otherwise, it is simply a random coin flip. The label of the point is then predicted according to the label that got the maximal number of votes. If a point is covered by a large

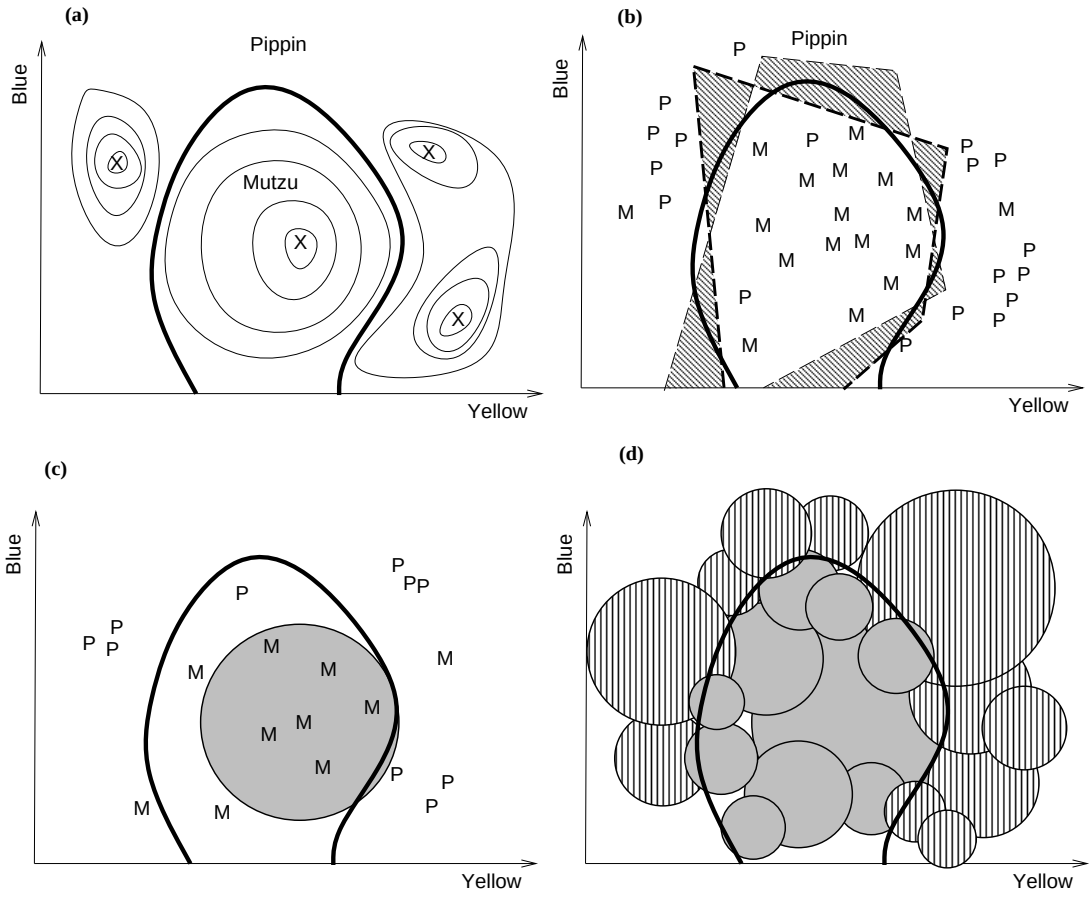


Figure 5.1: This figure describes the anticipated behavior of the boosting algorithm and the query algorithm on a real world learning problem. Part (a) describes a learning problem regarding the separation of two apple varieties according to their color. Each of the coordinates describes the intensity of a particular color. The bold line describes the optimal decision rule. The contour map defines the probability distribution of the apples, and $\times$'s denote the maximal density points. Part (b) describes the behavior of the query by committee algorithm. Each letter corresponds to a single training example. The two dotted lines describe two possible decision rules that have the same performance on the training examples. The query by committee algorithm accepts as queries those examples that fall in the highlighted areas. Part (c) describes a typical hypothesis chosen by the weak learner described in the text. The disk defines the hypothesis which predicts the label M inside the disk and predicts randomly outside it. Part (d) describes a possible set of weak hypotheses (overlapping disks) at a late point in the boosting process.

number of disks of the same type then it is likely to be labeled in the same way by the combined prediction rule.

As apples that have similar color are likely to be of the same type, it is reasonable that the algorithm will be able to find a large disk that contains many random instances with the same label. It is also likely that any new random instance that falls within this disk will have the same label. This means that the generated hypothesis is better than a random guess within the disk, which gives it an edge over a random guess of the label. However, it is clear that the size of this edge depends on the number of instances that fall within the disk, which, in turn, depends on the distribution of the instances. If most of the instances are of apples with ambiguous color, then instances with similar color are likely to be of different types, and the disk found by the learning algorithm is likely to be small and provide only a small edge over random guessing. It seems reasonable to assume that the distribution of actual apples gives high probability to colors that are far from the optimal decision border, as described in Figure 5.1(a), which implies that the hypothesis found by the boosting algorithm in the first stage is likely to have a large edge. However, as the boosting algorithm proceeds, it tends to accept examples close to the decision border, because these are the examples on which different hypotheses tend to disagree.

We thus expect that the behavior of the proposed learning algorithm will not fit within the standard weak-PAC learning framework, but rather be of the distribution-dependent type analyzed in Section 2.4.1. Using the results presented there we would like to show that if the edge of the weak learner does not decrease too rapidly as the distribution of the instances is changed by the boosting algorithm, then the boosting algorithm is able to generate an accurate hypothesis. However, several obstacles remain in the path to this type of analysis. First, we wish to consider *ambiguous* concepts, i.e. concepts that allow mapping a particular color to both types of apples. This type of mapping has been formalized by Kearns and Schapire [Kearns and Schapire, 1990] using the notion of *p-concepts*. However, little progress has been made so far on algorithms for boosting p-concepts[1] Second, we need to formalize the intuitive argument presented above regarding the dependence of the edge of the weak learner on the concentration of the instances around the border. This argument involves a close relationship between the hidden concept and the distribution of the instances, and such a relationship is completely outside the realm of the current theory of learning from random examples.

Let us now discuss using the query by committee algorithm (Chapter 3) in the context of the same problem. Suppose we have some reasonable learning algorithm that works for this kind of problem. For example, suppose that a neural network is capable of learning to separate the two varieties of apples according to their color. For simplicity, let us assume that the hypotheses computed by this network can be represented by regions bounded by a polygon with a small number of vertices. We can use the method of query by committee to try to reduce the number of instances that the human teacher needs to label. The idea is simple, instead of a single neural network, we train *two* neural networks. We use the same labeled examples to train both networks. The difference between the networks is a result of using a stochastic update rule that introduces small random perturbations into the training process so that each network

[1] Recently, Aslam and Decatur have shown how boosting can be used in the context of Kearns Statistical Queries model to boost weak learning algorithms in the context of independent noise on the labels.

arrives at a different set of weights. These perturbations also apply if the training error is zero, so that even in the stationary state the two networks perform a kind of a random walk in the space of hypotheses. Every unlabeled instance is presented to the two networks and they compute their predictions. If the two predictions differ, then the teacher is queried, and the labeled example is added to the training set. After each instance is presented, the two networks are trained, whether the example was added to the training set or not, to insure that a different pair of hypotheses is tried each time. The state of the algorithm after a number of training examples have been accumulated is described in Figure 5.1(b).

This algorithm seems to be a reasonable implementation of the query by committee method to this case. However, a rigorous analysis of its behavior is well beyond what we can currently achieve. Not only is the hidden concept a p-concept, but the hypotheses are taken from a different space than the hidden concepts. It is not clear whether random perturbations of the learning rule provide a good approximation of a Gibbs learning algorithm. Moreover, even if this is a reasonable approximation, it is not clear whether the equivalent prior distribution provides a reasonable approximation to the "correct" prior distribution that we assume is available to the learner in our analysis. However, if the qualitative results of our analysis carry over to this case, then the number of queries will be just a small fraction of all the random instances. Intuitively, that is clear, because after a small number of typical apples of each variety have been observed, each of the networks is likely to be correct on most of the typical instances. That intuition, which is illustrated in Figure 5.1(d), is a result of the fact that most of the examples are far from the borders. It is only the rare instances that are close to the border that can cause the algorithm to make a query. As the training set increases, both hypotheses become increasingly accurate and increasingly similar, and the frequency of queries decreases to zero.

Notice some similarities in the behavior of the two algorithms presented above. Both algorithms start by choosing examples at random from the space and then gradually concentrate on examples that are closer to the borders. As this concentration increases, more and more of the random examples are discarded, causing only a small fraction of the instances to take part in the actual learning. Also, in both of the algorithms the examples that are more important, and are usually accepted, are examples on which different hypotheses tend to disagree. In the current state of the theoretical analysis, these are only intuitive notions and there are many obstacles on the way to proving theorems that will formalize them. The obstacles are both technical and conceptual. On the technical level, a more sophisticated mathematical analysis is required to prove that results similar to the ones presented in this thesis exist in a broader context. On the conceptual level, we need mathematical frameworks for the analysis of learning that will incorporate prior assumptions about the structure of the world that are intuitively appealing but are ignored by the current frameworks.

A main assumption that was used in the discussion above is that the distribution of the instances is, in a strong sense, *helpful* to the learning process. This assumption is justified because in many cases both the instance and its label are generated by the same underlying process. In our apple classification example, both the color of the apple and its type are direct results of its genetic type. As a contrast, consider the task of classifying the two types of apples according to the bar-code number that is printed on their packages. In this case the bar-code, which defines the instance, is related mostly to the way in which packaging companies and shops are organized, and not to the genetic information of the apple. Consequently, we would not expect the distribution of these bar-codes to be related, in any simple way,

to the correct classification of the apples. Note that there exists a well-defined mapping from bar-codes to types, and this mapping might be learnable, however, in this case it seems clear that the learner can learn very little from unlabeled instances.

We believe that in many real-world learning problems the labels and the instances are both generated by the same mechanism and, as a result, the learning algorithm can assume a close relation between the distribution of the instances and the hidden target. The type of assumption on the distribution of the instances that we used in this section is that the more likely instances can be classified easily, while smaller and smaller parts of the instance space get harder and harder to classify. The work we present in Chapter 4 can be seen as a further step in this direction. Here we assume that we can deduce useful information from the instance distribution *alone*, without *any* label information. The work we present is concerned with a distribution model for high dimensional binary vectors, however, one of the basic motivations for this work can be demonstrated in our simple example of apple classification. In this problem, it is quite reasonable to assume that the distribution of the color of random apples tends to *cluster* around specific colors, which correspond to specific varieties of Mutzu and of Pippin. Unsupervised learning can be used to locate these clusters. It is then sufficient to query a teacher on one instance from each cluster to deduce the correct label of the whole cluster.

The experimental work we present at the end of Chapter 4 in which we try to learn the distribution of images of handwritten digits has a similar motivation. We believe that high-order correlations between the pixels in such images correspond to meaningful features such as lines and intersections, and that the images of typical digits are concentrated in clusters that correspond to different digits and can be described by combinations of such features. If this assumption is correct, then in order to learn to classify the images according to the digit that they contain, it is sufficient to obtain the correct classification of just one image from each cluster, greatly reducing the number of labeled instances required for learning.

It seems that the digit image data, which is very high-dimensional, is much more clustered then the apple color data. Indeed, we believe that in general, as the number of features that are used to characterize an instance increases, the distribution of the instances in feature space becomes increasingly concentrated around typical values and away from the optimal decision boundaries.

Other approaches to learning optimal prediction rules, such as pattern recognition [Duda and Hart, 1973a], incorporate the assumption that the distribution of the instances is informative directly into their basic mathematical framework. It is thus interesting to compare our approach to the approach used for learning in pattern recognition. We claim that our approach has advantages over the classical pattern recognition approach because while it is sensitive to the distribution of the instances, it ignores some properties of the distribution that are not relevant to making optimal predictions. We shall briefly describe a pattern recognition approach to this problem, and then compare it to the approaches described above.

The classical mathematical description of the apple variety prediction problem that is used in pattern recognition [Duda and Hart, 1973a] is the following. We assume that each of the two varieties of apples has some probability of having any particular color. Thus each variety of apple corresponds to a distribution over the plane. Nature generates a random example by selecting the variety of the apple at random and then selecting the associated color according to the corresponding distribution over the plane. The resulting distribution is a mixture of the two distributions that correspond to the two apple varieties.

Each color has some probability of corresponding to one of the two varieties, and the optimal prediction rule is to predict the variety that has a higher probability. One classical pattern recognition approach to learning the optimal decision rule is to first estimate the distribution of the colors of each variety of apple, and then to define the prediction rule by comparing the estimated probabilities that are associated with each color. The estimation of the distributions can be performed by either parametric or non-parametric methods [Duda and Hart, 1973a]. The parametric methods search for a distribution from a particular family of distributions, such as Gaussian mixtures, that best fits the data. Non-parametric methods, such as the k-nearest neighbor classification method, estimate the probabilities locally around each point of the input space. The approach that is suggested by our work is different than both the parametric and the non-parametric approaches in pattern recognition. Rather than estimating the distributions, we work on directly approximating the optimal prediction rule. Our methods are sensitive to the distribution of the instances in a different way than the pattern recognition methods. They start by gathering information on the decision rule from random examples and then, gradually, concentrate on those examples which are close to the border or have an atypical label.

This approach has an advantage over the pattern recognition approach when the distribution of the label that correspond to most of the color combinations is strongly biased to one of the two labels, while those color combinations on which the bias of the label distribution is weak are concentrated in small border areas. For the typical colors, very rough approximations of the distributions are sufficient for making the correct prediction. The pattern recognition approach ignores this property, while our algorithms use it to their advantage by concentrating on those instances whose label is more ambiguous.

To summarize, we think that the results presented in this thesis point to the important role that the distribution of instances plays in concept learning. In the popular distribution free learning model presented by Valiant, the input distribution is assumed to be the worst case. Another popular assumption is that the instance distribution is some specific distribution, such as the uniform distribution, a product distribution, or a mixture of Gaussians. We believe that in many cases one can assume that the instance distribution is closely related to the hidden target concept. While it is not clear what is the best way to formalize this type of assumption, it seems clear that such a formalization will make the problem of learning considerably easier and will bring computational learning theory closer to the practice of machine learning and pattern recognition.

References

[Ackley *et al.*, 1985] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[Angluin and Smith, 1983] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[Angluin, 1988a] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Angluin, 1988b] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Atkinson and Donev, 1992] A. C. Atkinson and A. N. Donev. *Optimum Experimental Designs.* Oxford science publications, 1992.

[Barland, 1992] Ian Barland. Some ideas on learning with directional feedback. Master's thesis, University of California at Santa Cruz, June 1992.

[Baum and Lang, 1991] E. B. Baum and K. Lang. Constructing hidden units using examples and queries. In *Advances in Neural Information Processing*, volume 3, 1991.

[Baum and Lang, 1992] E. B. Baum and K. Lang. Query learning can work poorly when a human oracle is used. In *International Joint Conference in Neural Networks*, Beijing, China, 1992.

[Baum, 1991] E. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, 2:5–19, 1991.

[Blumer *et al.*, 1986] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *18th ACM Symposium on Theory of Computing*, pages 273–282, Berkeley, 1986.

[Blumer *et al.*, 1987] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[Blumer *et al.*, 1989] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.

[Bollobas, 1985] B. Bollobas. *Random Graphs.* Academic Press, 1985.

[Bonnesen and Fenchel, 1987] T. Bonnesen and W. Fenchel. *Theory of Convex Bodies.* BCS Associates, Moscow, Idaho, USA, 1987.

[Cohn *et al.*, 1990] D. Cohn, L. Atlas, and R. Ladner. Training connectionist networks with queries and selective sampling. *Advances in Neural Information Processing Systems*, 2:566–573, 1990.

[Cox and Snell, 1989] D. R. Cox and E. J. Snell. *Analysis of binary data.* Chapman and Hall, 1989.

[Dempster *et al.*, 1977] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the *EM* algorithm. *Roy. Statist. Soc. B*, 39:1–38, 1977.

[Diaconis and Freedman, 1984] P. Diaconis and D. Freedman. Asymptotics of graphical projection pursuit. *Annals of Statistics*, 12:793–815, 1984.

[Drucker *et al.*, 1993] Harris Drucker, Robert E. Schapire, and Ptrice Simard. Improving performance in neural networks using a boosting algorithm. In *Proceedings of the fifth Conf. on Neural Informations Processing Systems*, San Mateo, CA, 1993. Morgan Kaufmann.

[Drucker, 1992 1993] H. Drucker. private correspondence, 1992–1993.

[Duda and Hart, 1973a] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[Duda and Hart, 1973b] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[Eisenberg and Rivest, 1990] Bonnie Eisenberg and Ronald L. Rivest. On the sample complexity of pac-learning using random and chosen examples. In *Proceedings of the 1990 Workshop on Computational Learning Theory*, pages 154–162, 1990.

[Everitt and Hand, 1981] B.S. Everitt and D.J. Hand. *Finite mixture distributions.* Chapman and Hall, 1981.

[Fedorov, 1972] V. V. Fedorov. *Theory of Optimal Experiments.* Academic Press, New York, 1972.

[Floyd and Warmuth, 1993] Sally Floyd and Manfred Warmuth. Sample compressions, learnability, and the vapnik-chervonenkis dimension. Technical Report UCSC-CRL-93-13, Computer and Information Sciences, University of California, Santa Cruz, 1993.

[Freeman, 1987] D. H. Freeman. *Applied Catagorical Data Analysis.* Marcel Dekker, 1987.

[Freund and Haussler, 1992] Y. Freund and D. Haussler. Unsupevised learning of distributions on binary vectors using two-layer networks. In *Proceedings of the 1991 Conf. on Neural Informations Processing Systems*, San Mateo, CA, 1992. Morgan Kaufmann.

[Freund *et al.*, 1993] Y. Freund, H.S Seung, E. Shamir, and N. Tishby. Accelerating learning using query by committee. In *Proceedings of the 1992 Conf. on Neural Informations Processing Systems (To appear)*, San Mateo, CA, 1993. Morgan Kaufmann.

[Freund, 1990] Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Workshop on Computational Learning Theory*, pages 202–216, San Mateo, CA, 1990. Morgan Kaufmann.

[Freund, 1992] Y. Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 391–398, San Mateo, CA, 1992. Morgan Kaufmann.

[Friedman and Tukey, 1974] J.H. Friedman and J.W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Comput.*, pages 881–889, 1974.

[Friedman *et al.*, 1984] J. H. Friedman, W.Stuetzle, and A. Schroeder. Projection pursuit density estimation. *J. Amer. Stat.Assoc.*, 79:599–608, 1984.

[Friedman, 1987] J. H. Friedman. Exploratory projection pursuit. *J. Amer. Stat.Assoc.*, 82(397):599–608, March 1987.

[Gefner and Pearl, 1987] Hector Gefner and Judea Pearl. On the probabilistic semantics of connectionist networks. Technical Report CSD-870033, UCLA Computer Science Department, July 1987.

[Geman and Geman, 1984] S Geman and D Geman. Stochastic relaxations, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6:721–742, 1984.

[Geman, 1986] Stuart Geman. Stochastic relaxation methods for image restoration and expert systems. In D.B. Cooper, R.L.Launer, and D.E. McClure, editors, *Automated Image Analysis: Theory and Experiments.* Academic Press, 1986.

[Gold, 1967] E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Goldmann *et al.*, 1992] M. Goldmann, J. Hastad, and A. Razborov. Majority gates vs. general weighted threshold gates. In *Structures conference Proceedings*, pages 2–13, 1992.

[Graham *et al.*, 1991] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics, a foundation for computer science.* Addison-Wesley, 1991.

[Haussler *et al.*, 1988] David Haussler, Nick Littlestone, and Manfred Warmuth. Predicting 0,1-functions on randomly drawn points. In *Proceedings of the 29th Annual Symposium on the Foundations of Computer Science*, pages 100–109. IEEE, 1988.

[Haussler *et al.*, 1991a] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95:129–161, 1991.

[Haussler *et al.*, 1991b] David Haussler, Michael Kearns, and Robert Schapire. Bounds on the sample complexity of bayesian learning using information theory and the vc dimension. In *Proceedings of the 1991 Workshop on Computational Learning Theory*, pages 61–74, San Mateo, CA, 1991. Morgan Kaufmann.

[Haussler *et al.*, to appear] D. Haussler, M. Kearns, and R. Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. *Machine Learning*, to appear.

[Hertz *et al.*, 1991] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction To The Theory Of Neural Computation*. Addison Wesley, 1991.

[Hopfield, 1982] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad Sci. USA*, 79:2554–2558, April 1982.

[Huber, 1985] P.J. Huber. Projection pursuit (with discussion). *Ann. Stat.*, 13:435–525, 1985.

[Jolliffe, 1986] I.T. Jolliffe. *Principle Component Analysis*. New York: Springer-Verlag, 1986.

[Kearns and Schapire, 1990] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *31st Annual Symposium on Foundations of Computer Science*, pages 382–391, 1990.

[Kearns and Valiant, 1988] M. Kearns and L.G. Valiant. Learning boolean formulae or finite automata is as hard as factoring. Technical Report TR-14-88, Harvard University Aiken Computation Laboratory, Cambridge, MA, 1988.

[Kearns and Valiant, 1989] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *21st ACM Symposium on Theory of Computing*, pages 433–444, Seattle, WA, 1989.

[Kearns, 1993] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the 25th ACM Symp. on Theory of Computing*, pages 392–401. ACM, 1993.

[Kharitonov, 1993] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th ACM Symp. on Theory of Computing*, pages 372–381. ACM, 1993.

[Kinzel and Ruján, 1990] W. Kinzel and P. Ruján. Improving a network generalization ability by selecting examples. *Europhys. Lett.*, 13:473–477, 1990.

[Lindley, 1956] D. V. Lindley. On a measure of the information provided by an experiment. *Ann. Math. Statist.*, 27:986–1005, 1956.

[Littlestone and Warmuth, 1986] Nick Littlestone and Manfred Warmuth. Relating data compression and learnability. This early and hard-to-locate work is referenced and partly re-written in FW93, 1986.

[Littlestone and Warmuth, 1989] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *30th Annual Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[McDiarmid, 1989] C. McDiarmid. On the method of bounded differences. In *Survey of Combinatorics, 10th British Combinatorial Conference*, 1989.

[Mitchell, 1978] Tom Mitchell. Version spaces: as approach to concept learning. Technical Report Tech. Report CS-78-711, Dept. of Computer Science, Stanford University, 1978.

[Natarajan, 1991] Balas K. Natarajan. *Machine Learning - a Theoretical approach*. Morgan Kaufmann, 1991.

[Neal, 1990] Radford M. Neal. Learning stochastic feedforward networks. Technical report, Department of Computer Science, University of Toronto, November 1990.

[Nowlan, 1990] S. Nowlan. Maximum likelihood competitive learning. In D. Touretsky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 574–582. Morgan Kaufmann, 1990.

[Oja, 1989] E. Oja. Neural networks, principle components, and subspaces. *Int. J. Neural Systems*, 1(1):61–68, 1989.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[Peterson and Anderson, 1987] Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[Poggio and Girosi, 1989] Tomaso Poggio and Federico Girosi. A theory of networks for approximation and learning. Technical Report A.I. Memo No. 1140, Massachusetts Institute of Technology, Cambridge, MA, 1989.

[Rabiner and Juang, 1986] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[Rissanen, 1986] Jorma Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.

[Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, Mass., 1986.

[Sanger, 1989] T.D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.

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

[Schapire, 1990] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–226, 1990.

[Schapire, 1991] Robert E. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. PhD thesis, M.I.T., 1991.

[Schapire, 1992] Robert E. Schapire. private correspondence, January 1992.

[Serfling, 1980] R. J. Serfling. *Approximation Theorems of Mathematical Statistics*. John Wiley & Sons, 1980.

[Seung *et al.*, 1992] H.S Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294, San Mateo, CA, 1992. Morgan Kaufmann.

[Shamir, 1992] E. Shamir. private correspondence, 1992.

[Smith, 1985] Peter Smith. *Convexity Methods in Variational Calculus*. Research studies press, John Wiley & sons, 1985.

[Turán, 1993] G. Turán. Lower bounds for pac learning with queries. In *Proceedings of the sixth Workshop on Computational Learning Theory*, pages 384–391, 1993.

[Valiant, 1984a] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–42, 1984.

[Valiant, 1984b] L. G. Valiant. A theory of the learnable. *Comm. ACM*, 27:1134–1142, 1984.

[Vapnik, 1982] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

Appendix A. Appendixes regarding Boosting by Majority

A.1 Boosting the reliability of a learning algorithm

We present the boosting algorithm, $\mathbf{B}_{\mathrm{Rel}}$, in Figure A.1, and prove its performance.

Proof of Lemma 2.3.6 The bound on the number of examples is immediate from the definition of the algorithm. To prove that the algorithm is correct, we bound the probability that the resulting hypothesis has error larger than $1/2 - \gamma/2$. There are two events that might cause this. The first is that all of the r hypotheses generated by **WeakLearn** have error larger than $1/2 - \gamma$. The second is that a hypothesis that has error larger than $1/2 - \gamma/2$ makes less mistakes, on the test sample, than a hypothesis that has error smaller than $1/2 - \gamma$. It is easy to bound the probability of each of those events by $\delta/2$. Which proves the lemma.

As we know that each call to **WeakLearn** has probability of at least λ of generating a hypothesis with error smaller than $1/2 - \gamma$ at each trial, the probability of not generating any accurate enough hypothesis is at most

$$(1-\lambda)^r = (1-\lambda)^{1/\lambda \ln(2/\delta)} \leq e^{-\ln(2/\delta)} = \delta/2 \ .$$

In order for the second event to happen, given that one of the hypotheses has error smaller than $1/2 - \gamma$, there has to be a bad hypothesis whose estimated error is larger than that of the good hypothesis. For this to happen, the gap between the actual error and the estimated error for at least one of the r hypotheses has to be at least $\gamma/4$. Using Hoeffding bounds we get that this probability is at most

$$re^{-2m(\gamma/4)^2} = r\exp(-2(8/\gamma^2)\ln(2r/\delta)(\gamma/4)^2) = re^{-\ln(2r/\delta)} = \delta/2 \ ,$$

which proves the lemma. ∎

A.2 Divisibility lemma

Lemma A.2.1: *If the probability space $\langle X, \Sigma, V\rangle$ is divisible, then, for any set $D \in \Sigma$ there exists a set $G \subset D, G \in \Sigma$ such that $V(G) = (1/2+\gamma)V(D)$ and $W(G) \geq (1/2+\gamma)W(D)$*

Algorithm $\mathbf{B}_{\mathrm{Rel}}$
Input: **EX**,**WeakLearn**, γ, λ, δ
Output: A hypothesis h_M, that has error smaller than $1/2 - \gamma/2$ with probability at least $1 - \delta$.

1. Call **WeakLearn** $r = \frac{\ln(2/\delta)}{\lambda}$ times, each time on a different set of random examples. Store the resulting hypotheses as $h_1, \ldots, h_r$.
2. Count the number of mistakes made by each of the r hypotheses on a random sample of size $m = (8/\gamma^2)\ln(2r/\delta)$.
3. Return the hypothesis that makes the smallest number of mistakes on the sample.

Figure A.1: A description of the algorithm for boosting the reliability of an algorithm.

Proof: First observe that as we are interested only in the ratio of the weight and value of G to that of D, so w.l.o.g. we can assume that $V(D) = W(D) = 1$.

Define the following series of partitions of D.

- $\mathcal{P}_0 = \{D\}$.
- $\mathcal{P}_1 = \{D_0^1, D_1^1\}$ where the sets are disjoint and $V(D_0^1) = V(D_1^1) = \frac{1}{2}$.
- Construct $\mathcal{P}_{i+1}$ from $\mathcal{P}_i$ by splitting each set in $\mathcal{P}_i$ into two equal valued parts. So that the value of each part is exactly 2^{-i}.

We shall now use the partitions $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ to construct a series of sets $G_0, G_1, G_2, \ldots$ that will provide better and better approximations of the target set G. Assume that the binary expansion of $1/2+\gamma$ is

$$\frac{1}{2} + \gamma = \sum_{j=0}^{\infty} b_j 2^{-j}$$

(note that $b_0 = 0, b_1 = 1$) and construct the sets G_i according to the following inductive procedure:

$$G_0 = \emptyset$$

$$\Delta_i = \text{ the set with the largest value in } \mathcal{P}_i \text{ that is not a subset of } G_{i-1}$$

$$G_{i+1} = \begin{cases} \text{if } b_i = 0 & G_i \\ \text{if } b_i = 1 & G_i \cup \Delta_i \end{cases}$$

It is clear that G_i is a monotonically increasing series of sets and that $\lim_{i\to\infty} V(G_i) = 1/2 + \gamma$. Also clearly from the way Δ_i is chosen we have

$$\frac{W(\Delta_i)}{1 - W(G_i)} \geq \frac{V(\Delta_i)}{1 - V(G_i)}$$

We shall now prove by induction on i that $\forall i \geq 0 \;\; W(G_i) \geq V(G_i)$.

- For $i = 0$, $G_0 = \emptyset$ so the claim holds trivially.
- For $i \geq 1$, if $b_i = 0$ then $G_{i+1} = G_i$ so the induction holds trivially. Else, $b_i = 1$ and thus $G_{i+1} = G_i \cup \Delta_i$ and we get:

$$W(G_{i+1}) = W(G_i) + W(\Delta_i) = V(G_i) + (W(G_i) - V(G_i)) + W(\Delta_i) \geq$$

$$V(G_i) + (W(G_i) - V(G_i)) + V(\Delta_i)\frac{1 - V(G_i) - (W(G_i) - V(G_i))}{1 - V(G_i)} =$$

$$V(G_i) + V(\Delta_i) + (W(G_i) - V(G_i))[1 - \frac{1}{1 - V(G_i)}]$$

The first two terms sum to $V(G_{i+1})$, and the last term is positive because $V(G_i) \leq 1$ and from the induction hypothesis $W(G_i) - V(G_i) > 0$, the induction hypothesis is thus proven.

Define $G = \bigcup_{j=1}^{\infty} G_i$. As all Δ_i are in the sigma algebra Σ then so is G. Also $V(G) = \lim_{i\to\infty} V(G_i) = 1/2 + \gamma$. similarly $W(G) = \lim_{i\to\infty} W(G_i) = 1/2 + \gamma$ and because for all i we have $W(G_i) \geq V(G_i)$, we also get an inequality at the limit $W(G) \geq V(G) = 1/2 + \gamma$, which proves the lemma. ∎

A.3 Proof of Lemma 2.3.10

In order to prove the lemma, we use the following technical lemma:

Lemma A.3.1: *For any real numbers $x \geq 1$ and $0 \leq \mu < 1/2$*

$$\exp\left(-\frac{1}{3(1-4\mu^2)x}\right) < \frac{\binom{x}{x(1/2-\mu)}}{\sqrt{\frac{2}{\pi x}}e^{xH(1/2-\mu)}} < \frac{\exp\left(\frac{1}{12x}\right)}{\sqrt{1-4\mu^2}} \ . \tag{A.1}$$

Where $H(y) = -y\ln y - (1-y)\ln(1-y)$ is the entropy function, and the extension of the binomial function to the reals is based on the extension of the factorial to the Gamma function $x! = ?\,(x+1)$.

Proof: The proof of this lemma is based on the Stirling approximation. Notice that as $x \to \infty$, the lower bound converges to 1 while the upper bound converges to $(1-4\mu^2)^{-1/2}$. In other words, for large values of x the binomial $\binom{x}{x(1/2-\mu)}$ is related to the exponential function in the denominator by a small factor.

Stirling approximation to the factorial can be written in the following way:[1]

$$\forall\ x \geq 1\ \ x\ln x - x + \frac{\ln x}{2} + \ln\sqrt{2\pi} < \ln(x!) < x\ln x - x + \frac{\ln x}{2} + \ln\sqrt{2\pi} + \frac{1}{12x} \ .$$

From which we get the lower bound as follows

$$\begin{aligned}
\ln\binom{x}{x(1/2-\mu)} &= \ln x! - \ln((1/2-\mu)x)! - \ln((1/2+\mu)x)! \\
&> x\ln x - x + \frac{\ln x}{2} + \ln\sqrt{2\pi} \\
&\quad -x(1/2-\mu)\ln(x(1/2-\mu)) + x(1/2-\mu) - \frac{\ln(x(1/2-\mu))}{2} - \ln\sqrt{2\pi} - \frac{1}{12(1/2-\mu)x} \\
&\quad -x(1/2+\mu)\ln(x(1/2+\mu)) + x(1/2+\mu) - \frac{\ln(x(1/2+\mu))}{2} - \ln\sqrt{2\pi} - \frac{1}{12(1/2+\mu)x} \\
&= xH(1/2-\mu) - \ln\sqrt{x} - \ln\sqrt{\frac{1}{4}-\mu^2} - \ln\sqrt{2\pi} - \frac{1}{12(1/4-\mu^2)x} \\
&\geq xH(1/2-\mu) - \ln\sqrt{2\pi x} + \ln 2 - \frac{1}{3(1-4\mu^2)x} \ .
\end{aligned}$$

And the upper bound as follows

$$\begin{aligned}
\ln\binom{x}{x(1/2-\mu)} &= \ln x! - \ln((1/2-\mu)x)! - \ln((1/2+\mu)x)! \\
&< x\ln x - x + \frac{\ln x}{2} + \ln\sqrt{2\pi} + \frac{1}{12x} \\
&\quad -x(1/2-\mu)\ln(x(1/2-\mu)) + x(1/2-\mu) - \frac{\ln(x(1/2-\mu))}{2} - \ln\sqrt{2\pi} \\
&\quad -x(1/2+\mu)\ln(x(1/2+\mu)) + x(1/2+\mu) - \frac{\ln(x(1/2+\mu))}{2} - \ln\sqrt{2\pi} \\
&= xH(1/2-\mu) - \ln\sqrt{x} - \ln\sqrt{\frac{1}{4}-\mu^2} - \ln\sqrt{2\pi} + \frac{1}{12x} \\
&= xH(1/2-\mu) - \ln\sqrt{2\pi x} + \ln 2 - \ln\sqrt{1-4\mu^2} + \frac{1}{12x} \ .
\end{aligned}$$

[1] See, for example, Equation (9.91) in [Graham *et al.*, 1991].

∎

Proof of Lemma 2.3.10 : We can rewrite the definition of α_r^i from Figure 2.4 as follows (ignoring the choices of r that give $\alpha_r^i = 0$ for the purpose of the upper bound):

$$\alpha_i^r = \binom{x}{(\frac{1}{2}-\mu)x}(\frac{1}{2}-\frac{\gamma}{2})^{x(\frac{1}{2}-\mu)}(\frac{1}{2}+\frac{\gamma}{2})^{x(\frac{1}{2}+\mu)} ,$$

where $x = k-i-1$ and $\mu = 1/2 - (\lfloor k/2 \rfloor - r)/(k-i-1)$. Using the upper bound given in Lemma A.3.1 bound the last expression for any value of μ

$$\binom{x}{(\frac{1}{2}-\mu)x}(\frac{1}{2}-\frac{\gamma}{2})^{x(\frac{1}{2}-\mu)}(\frac{1}{2}+\frac{\gamma}{2})^{x(\frac{1}{2}+\mu)}$$

$$< \sqrt{\frac{2}{\pi x}}\frac{e^{1/12x}}{\sqrt{1-\gamma^2}}\exp\left(xH(\frac{1}{2}-\mu) + x(\frac{1}{2}-\mu)\ln(\frac{1}{2}-\frac{\gamma}{2}) + x(\frac{1}{2}+\mu)\ln(\frac{1}{2}+\frac{\gamma}{2})\right) .$$

But a basic inequality is that for any $-1/2 \leq \mu \leq 1/2$

$$H(\frac{1}{2}-\mu) \leq -(\frac{1}{2}-\mu)\ln(\frac{1}{2}-\frac{\gamma}{2}) - (\frac{1}{2}+\mu)\ln(\frac{1}{2}+\frac{\gamma}{2}) .$$

Where equality is achieved only when $\mu = \gamma/2$. From this we get that

$$\binom{x}{(\frac{1}{2}-\mu)x}(\frac{1}{2}-\frac{\gamma}{2})^{x(\frac{1}{2}-\mu)}(\frac{1}{2}+\frac{\gamma}{2})^{x(\frac{1}{2}+\mu)} < \sqrt{\frac{2}{\pi x}}\frac{e^{1/12x}}{\sqrt{1-\gamma^2}} .$$

As $\gamma \leq 1/2$, and $x \geq 1$, we get the statement of the lemma. ∎

Appendix B. Projection distributions of the binary combination model.

In this section we use results from [Diaconis and Freedman, 1984] to show that the projections of the binary combination model are very similar to those of the real-valued combination model model when the weight vectors are small. As has been discussed in Section (4.2.3), the binary combination model distribution can be viewed as a mixture of 2^m generalized binomial distributions. We call these binomial distributions *binoms*. Each binom corresponds to a particular setting of the hidden vector $\vec{h}$ and to a single Gaussian component in the real-valued model. We shall show that although the distribution of the binoms are very different from the corresponding Gaussians, their projections onto almost any direction are very similar. This implies that the projections of the binary-valued combination model are very similar to those of the real-valued combination model. Because Projection pursuit methods depend only on properties of the projections of the distribution, it is a valid approximation to use the real-valued combination model for learning distributions generated by a binary-valued combination model.

The mixture coefficients of the binoms are $Pr(\vec{h}|\phi)$ as defined in Equation (4.13). The mean of the binom corresponding to $\vec{h}$ is $\mu(\vec{h}_i) = \tanh(\sum_{i=1}^{m} h_i\omega^{(i)})$ where by $\tanh(\vec{x})$ we denote the application of tanh to each component of $\vec{x}$. If the weight vectors $\omega^{(i)}$ are all small then $\tanh(\sum_{i=1}^{m} h_i\omega^{(i)}) \approx \sum_{i=1}^{m} h_i\omega^{(i)}$, and we get that the means of the binoms are very close to the means of the corresponding Gaussians. Next we show that under mild assumptions, the projection of each binom is very close to a Gaussian.

Diaconis and Freedman [Diaconis and Freedman, 1984] discuss conditions under which most projections of high-dimensional data sets are close to Gaussian. Their analysis considers large sets of points taken from high dimensional spaces. These points are not assumed to be generated by a distribution. Instead, the conditions for Gaussianity of the projection are given as geometric relations among the points. These relations must hold in the limit where both the dimension of the space and the size of the sets tends to infinity. We shall show that if the weight vectors of the combination model are generated by some distribution then, with high probability, samples generated by each binom have the required geometric properties and thus most of their projections are close to Gaussians.

We follow most of the notation used in [Diaconis and Freedman, 1984]. Let $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$ be vectors in R^n, this is the data set. Suppose that n, N and the data set all depend on some common index ν, and that as ν tends to infinity, so do n and N. Let S_{n-1} be the unit sphere in R^n and let γ be chosen uniformly at random from S_{n-1}. Theorem 1.1 in [Diaconis and Freedman, 1984] states that if the following conditions hold, then the empirical distribution of $\gamma \cdot \vec{x}_i$ converges weakly to the normal distribution $\mathcal{N}(0, \sigma^2)$ in probability, as $\nu \to \infty$. Where "weak convergence" is convergence as a measure on R and "in probability" is w.r.t. the uniform distributions on S_{n-1}.

The required conditions follow. There must exist some finite and positive σ^2 such that for any positive ϵ, the following limits hold as ν tends to infinity,

$$\left| \{1 \le j \le N : |\|\vec{x}_j\|_2^2 - \sigma^2 n| > \epsilon n\} \right| / N \to 0 \tag{B.1}$$

$$|\{1 \le j,k \le N : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n\}| / N^2 \to 0 \tag{B.2}$$

Where # denotes the cardinality of a set. The first condition intuitively means that vectors are almost all of almost the same length. The second condition means that most pairs of vectors are close to orthogonal.

We are interested in projections of samples generated by the combination model, as these are random samples, we would like to show that the geometric conditions hold with probability one. Suppose we have a sequence of binomial distributions over binary cubes of increasing dimension: $\{-1,+1\}, \{-1,+1\}^2, \ldots, \{-1,+1\}^n, \ldots$. Each distribution is fully specified by its mean vector: $\vec{\mu}_1 \in [-1,+1], \vec{\mu}_2 \in [-1$ Suppose that we have a sample from each distribution and that the sample size increases with the dimension n of the space: $\langle \vec{x}_1^1 \rangle, \langle \vec{x}_1^2, \vec{x}_2^2 \rangle, \cdots, \langle \vec{x}_1^n, \ldots, \vec{x}_n^n \rangle, \cdots$. We would like to show that random projections of these samples produce empirical marginal distributions that are very close to Gaussian distributions with a probability that goes to 1 as $n \to \infty$. However, it is not hard to construct sequences of mean vectors such that this will not happen. For instance, if $\vec{\mu} = \{0, +1, \ldots, +1\}$, then the distribution is concentrated in the two points $\{-1, +1, \ldots, +1\}$, and $\{+1, +1, \ldots, +1\}$, and all projections of this distribution will also be concentrated on two points.

We prove that the desired asymptotic conditions hold with probability 1 if the mean vectors $\vec{\mu}_n$ are selected in the following way. Assume there is some distribution $\bar{P}$ on $[-1,+1]$ and that each component of each $\vec{\mu}_n$ is drawn independently at random from this distribution. For this to hold for the mixture components of the combination model it is enough to assume that the components of the weight vectors in the model underlying the data are chosen independently at random.

Theorem B.0.2: *Suppose that a sequence of vectors of increasing dimension:*

$$\vec{\mu}_1 \in [-1,+1], \vec{\mu}_2 \in [-1,+1]^2, \ldots, \vec{\mu}_n \in [-1,+1]^n, \ldots$$

is randomly drawn by selecting each component of each vector according to some distribution $\bar{P}$ over $[-1,+1]$.

Each vector $\vec{\mu}_n$ defines a distribution over $\{-1,+1\}^n$ in which the components are independent and the expected value is $\vec{\mu}_n$. Suppose that for each n we draw n vectors from this distribution, and that from each random vector we subtract the mean, $\vec{\mu}_n$.

Suppose that for each n we draw a vector $\vec{w}$ uniformly at random from the n dimensional unit sphere, project the n random vectors on the direction defined by $\vec{w}$ and assign each of the points in the projection a probability mass of $1/n$. In this way we create, for each n, a discrete distribution over the reals.

With probability one, over all the random choices that create the sequence of distributions, there exists $\sigma \geq 0$ such that the sequence of distributions converges weakly to the normal distribution $\mathcal{N}(0, \sigma^2)$.[1]

Proof: We prove the theorem by showing that the conditions of Theorem 1.1 in [Diaconis and Freedman, 1984] hold with probability one.

The proof of the condition B.1 is a simple application of the Markov bound. We wish to show that for some σ and for any $\epsilon, \delta > 0$:

$$\lim_{n \to \infty} P(\#\{1 \leq j \leq n : |\|\vec{x}_j\|_2^2 - \sigma^2 n| > \epsilon n\} > \delta n) = 0$$

[1] Weak convergence means that for any measurable set A, the probability assigned to A by the sequence of distributions converges to the probability of the limit distribution.

The n examples are independent, thus as n increases the fraction of the vectors that obey the condition becomes very close to the probability of obeying the condition. Thus it suffices to show that for a randomly chosen example $\vec{x}$

$$\lim_{n\to\infty} P(\ \|\vec{x}\|_2^2 - \sigma^2 n| > \epsilon n) = 0$$

the squared length of a vector is a sum of the squares of its components. As the components are chosen independently at random according to the mean vector $\vec{\mu}_n$ and as the components of $\vec{\mu}_n$ are chosen independently at random according to $\bar{P}$ we get that the average length of $\vec{x}$ is $n(1 - \int_{-1}^{+1} x^2 d\bar{P}(x))$. The variance of each term is at most 1. Thus defining σ^2 to be $1 - \int_{-1}^{+1} x^2 d\bar{P}(x)$ and using Markov bounds we get that

$$P(\ \|\vec{x}\|_2^2 - \sigma^2 n| > \epsilon n) \leq \frac{n}{(\epsilon n)^2} = \frac{1}{n\epsilon^2}.$$

and as n increases the probability decreases to zero as desired.

The proof of condition (B.2) is a bit more involved, because in this case the n^2 pairs that are checked for the condition are not independent. However, using the theory of U-statistics [Serfling, 1980][Chap. 5] their behavior can be related to that of independently drawn pairs. We wish to show that for any $\epsilon, \delta > 0$:

$$\lim_{n\to\infty} P(\#\{1 \leq j,k \leq N : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n\} > \delta n^2) = 0$$

first observe that when $j = k$ the condition will most often not hold, as we have just proved that the squared length of a vector is concentrated around $\sigma^2 n$. However we can ignore this set as it is a vanishing fraction of the n^2 pairs. It is thus sufficient to prove that

$$\lim_{n\to\infty} P(\#\{1 \leq j,k \leq n;\ j \neq k : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n\} > \delta n(n-1)) = 0$$

Using the notation of [Serfling, 1980] we define

$$h(\vec{x},\vec{y}) = \begin{cases} 1 & \text{if } |\vec{x}\cdot\vec{y}| > \epsilon n \\ 0 & \text{otherwise} \end{cases}$$

and observe the corresponding U-statistic, that is a random variable defined over samples of size n:

$$U(\vec{x}_1,\ldots,\vec{x}_n) = \frac{2}{n(n-1)} \sum_{1\leq i<j\leq n} h(\vec{x}_i,\vec{x}_j)$$

This random variable is exactly the cardinality of the set of pairs that have a dot product larger than ϵn divided by $n(n-1)$. Our goal is thus reduced to proving that the probability of a sample for which U is too large is small. We do that by using Markov inequality. The fact that U is an unbiased statistic means that the average of U is equal to the average of $h(\vec{x},\vec{y})$ when $\vec{x}$ and $\vec{y}$ are chosen independently at random. In other words it is equal to the probability that two randomly chosen vectors have a dot product larger than ϵn. We shall denote that probability by t. The variance of U can be related to the variance of $h(\vec{x},\vec{y})$ by using Lemma A. from page 183 of [Serfling, 1980].

$$Var(U(\vec{x}_1,\ldots,\vec{x}_n)) \leq \frac{2}{n(n-1)}[2(n-2)\zeta_1 + \zeta_2] \leq \frac{4}{n}\zeta_2$$

Where ζ_2 is simply the variance of $h(\vec{x}, \vec{y})$ when $\vec{x}$ and $\vec{y}$ are chosen independently at random. As $h(\vec{x}, \vec{y})$ is either 0 or 1, its variance is $t(1-t)$. Putting the bound on the variance into the Markov bound we get:

$$P\left[n(n-1)U(\vec{x}_1, \ldots, \vec{x}_n) > \delta(n(n-1))\right] \leq P\left[|U(\vec{x}_1, \ldots, \vec{x}_n) - t| > \delta - t\right] \leq \frac{4}{n}\frac{t(1-t)}{(\delta - t)^2}$$

It is easy to see that

$$t = P(|\vec{x} \cdot \vec{y}| > \epsilon n) \leq \frac{4}{\epsilon^2 n}$$

thus $\lim_{n\to\infty} t = 0$ and we get that the desired probability goes to zero, which completes the proof. ∎