A Generalization Model and Learning in Hardware

Alexander Nicholson
Department of Computer Science
California Institute of Technology
Pasadena, CA, 91125
zander@cs.caltech.edu

Abstract

We study two problems in the field of machine learning. First, we propose a novel theoretical framework for understanding learning and generalization which we call *the bin model*. Using the bin model, a closed form is derived for the generalization error that estimates the out-of-sample performance in terms of the in-sample performance. We address the problem of overfitting, and show that using a simple exhaustive learning algorithm it does not arise. This is independent of the target function, input distribution and learning model, and remains true even with noisy data sets. We apply our analysis to both classification and regression problems and give an example of how it may be used efficiently in practice. Second, we investigate the use of learning and evolution in hardware for digital circuit design. Using the reactive tabu search for discrete optimization, we show that we can learn a multiplier circuit from a set of examples. The learned circuit makes less than 2% error and uses fewer chip resources than the standard digital design. We compare use of a genetic algorithm and the reactive tabu search for fitness optimization. and show that the reactive tabu search performs significantly better on a 2-bit adder design problem for a similar execution time.

CaltechCSTR/2000.007

Submitted November 21, 2000

Acknowledgements

Several people deserve to be acknowledged for helping make this thesis possible.

The work on the bin model presented in chapter 2 is built largely upon work done by Xubo Song, and was done with much helpful input from Malik Magdon-Ismail.

Arrigo Benedetti and Roberto Battiti were invaluable in helping me get started on the hardware learning project, and Pietro Perona has provided helpful suggestions along the way.

Most importantly, I thank my advisor, Yaser Abu-Mostafa for assistance and guidance of all types throughout my research and studies. I'd also like to thank Joe Sill, Amir Atiya and Zehra Cataltepe for many useful meetings and discussions.

Finally, I thank my parents for their love and support.

This work was supported in part by the Center for Neuromorphic Systems Engineering (a National Science Foundation supported Engineering Research Center) under National Science Foundation Cooperative Agreement EEC 9402726.

Abstract

We study two problems in the field of machine learning. First, we propose a novel theoretical framework for understanding learning and generalization which we call *the bin model*. Using the bin model, a closed form is derived for the generalization error that estimates the out-of-sample performance in terms of the in-sample performance. We address the problem of overfitting, and show that using a simple exhaustive learning algorithm it does not arise. This is independent of the target function, input distribution and learning model, and remains true even with noisy data sets. We apply our analysis to both classification and regression problems and give an example of how it may be used efficiently in practice. Second, we investigate the use of learning and evolution in hardware for digital circuit design. Using the reactive tabu search for discrete optimization, we show that we can learn a multiplier circuit from a set of examples. The learned circuit makes less than 2% error and uses fewer chip resources than the standard digital design. We compare use of a genetic algorithm and the reactive tabu search for fitness optimization. and show that the reactive tabu search performs significantly better on a 2-bit adder design problem for a similar execution time.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Learning from examples has become a common technique for dealing with unstructured or mathematically ill-defined problems (see for instance [Duda and Hart, 1973] [Bishop, 1995a] [Haykin, 1994] [Vapnik, 1998] for an introduction). The task at hand is to extract relevant information from a finite set of examples to develop predictive models. In this thesis we address two important issues related to machine learning. First we study the question of generalization, that is, how well we can expect our learned system to generalize to data it has not encountered in the learning process. Second, we address the practical problem of learning in hardware, allowing much greater learning and evaluation speeds. In order to facilitate the discussion, we begin with an introduction of the general learning process and much of the notation and terminology used in the rest of the thesis.

1.1 The Learning Problem

In a learning problem, we are presented with a data set $\mathcal{D}$, the *training set*, which consists of N input-output pairs $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$. The input-output pairs are generated according to an *unknown* underlying function $f : \mathcal{X} \rightarrow \mathcal{Y}$, which we call the *target function*. Each $x_i \in \mathcal{X}$ is drawn from some input probability measure $p_X(x)$. The training set is our sole knowledge pertaining to the target function, and our goal is to infer the target function based on the training set. Learning entails choosing a hypothesis function $g : \mathcal{X} \rightarrow \mathcal{Y}$ from a collection of candidate functions $\mathcal{G}$ as our inference of the target function. The set $\mathcal{G}$ is called the *learning model* because it reflects how we choose to model the target function. The hypothesis function is chosen by a *learning algorithm*, $\mathcal{A}$. The learning algorithm takes as inputs the training set $\mathcal{D}$ and the learning model $\mathcal{G}$, and outputs a hypothesis g, usually based on some performance criterion on the training set. For example, a typical learning algorithm

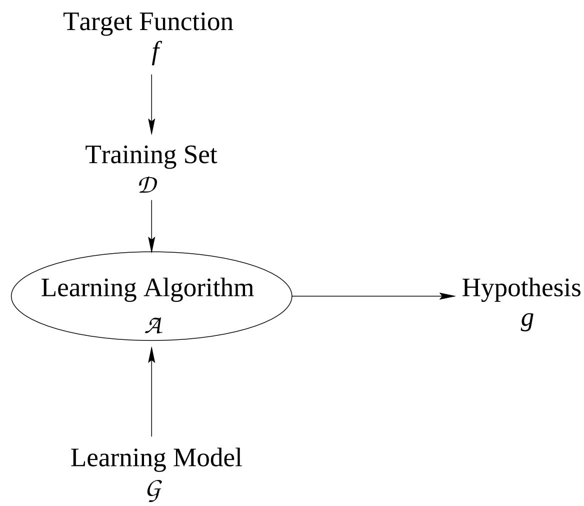
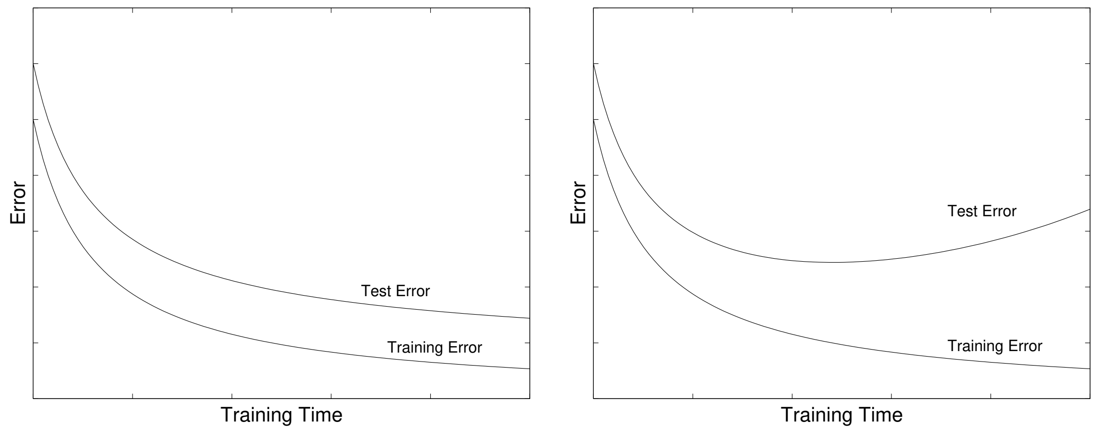


Figure 1.1: The learning process

might be to choose from the learning model the hypothesis which minimizes an error measure on the training set.

We measure how well the hypothesis g matches the target function f based on some error measure $e : \mathcal{G} \times \mathcal{X} \to \mathbf{R}$. Error occurs when a hypothesis deviates from the target function on any point in the input space. The error a hypothesis function commits on the training set is referred to as the *in-sample* or *training error*, and the error on points out of the training set is termed the *out-of-sample* or *test error*. A good hypothesis should give low out-of-sample error. A "clever" learning algorithm might be able to find a hypothesis that fits the training set well and achieves a small in-sample error.

Typically, two possible scenarios may occur in a learning process as illustrated in Figure 1.2. In the first scenario, the out-of-sample error decreases as the in-sample error decreases, all the way until the in-sample error reaches its minimum. In this case, it is good for a learning algorithm to find a hypothesis that achieves the minimum in-sample error, since the corresponding out-of-sample error is also minimal. In the second scenario, the out-of-sample error decreases at the beginning together with the in-sample error as the learning algorithm is getting a grasp of the general properties of the target function contained in the training set. Then at some point, the out-of-sample error starts to rise as the in-sample error is further reduced. The learning



(a) (b)

Figure 1.2: Two learning scenarios, one with overfitting

algorithm is finding hypotheses that fit the training data better, but that also fit any idiosyncrasies in the training set. As a result, the hypothesis approximates the training data set well, but fails to generalize to out-of-sample data. This phenomenon is usually referred to as *overfitting*. Overfitting occurs when the learning model is overly complex and is especially prominent when the training set is noisy.

This raises some important questions: When can we be confident that our model has truly learned and not simply memorized the examples it is given? How much does the in-sample error tell us about the out-of-sample error? This is the issue of *generalization*.

1.2 Generalization

Once we have selected a hypothesis g to model our target function, we would like to know how well it will perform on data not necessarily in the training set (out-of-sample data). That is, we want to know if the hypothesis can *generalize* from the training set to the entire space $\mathcal{X}$. We would like to be able to make quantitative statements about the out-of-sample performance. The problem is that we only have access to the training error, and if we have no constraints on the target function, the in-sample error may have no bearing on the out-of-sample error [Wolpert, 1996].

A great deal of effort has been made to assess the generalization performance of a learning process. The Prediction Error [Akaike, 1970] [Moody, 1991], VC analysis [Vapnik and Chervonenkis, 1971] [Abu-Mostafa, 1989], and Exhaustive Learning paradigm [Solla, 1992] [Schwartz *et al.*, 1990] are some of the significant related results.

The Prediction Error approach gives a general form for the out-of-sample error that is the sum of two terms:

$$\text{out-of-sample error} \approx \text{in-sample error} + \frac{2C}{N}\sigma^2 \tag{1.1}$$

where C embeds model complexity. The second term can be viewed as a penalty for more complex models. σ^2 is the variance of noise in the data. This criterion has the nice property of formulating the impact of model complexity and data noise on the gap between in- and out-of-sample error. The prediction error criterion is an asymptotic result, and for the nonlinear case [Moody, 1991], it is assumed that the input density is discrete with support only at input points. These are required for mathematical feasibility, but may limit the accuracy of the approximation in practice.

Some insights into generalization are gained by bounding the deviation between training and test error in the worst-case scenario. VC theory provides exactly such a bound:

$$P[\sup_{g \in \mathcal{G}} |\pi_g - \nu_g| < \varepsilon] < \delta(\varepsilon, N, d_{VC})$$

where ε is a tolerance level for the deviation between in-sample error ν and out-of-sample error π for a hypothesis, N is the number of training examples available and d_{VC} is a parameter related to the model complexity (the VC-dimension). This criterion highlights the impact of number of training examples N and model complexity on generalization. Because it is a worst-case analysis, the VC bound can be applied universally. This is not without drawbacks, though, since in practical scenarios the bound is often found to be weak. Also, the confidence level δ is a function of the *growth function* [Abu-Mostafa, 1989], the calculation of which can be formidable for general learning models.

Schwartz et al. [Schwartz *et al.*, 1990] suggested a framework with a somewhat similar formulation to our model, which they call *exhaustive learning*. Their work leads to the following result:

$$G_N = \int_0^1 g\rho_N(g)dg \tag{1.2}$$

$$= \frac{\int_0^1 g^{N+1}\rho_0(g)dg}{\int_0^1 g^N\rho_0(g)dg} \tag{1.3}$$

G_N is termed the 'mean generalization ability', and we will see that it effectively corresponds to $1 - E[\pi]$ in our analysis. Similarly, $1 - g$ is equivalent to our definition of π, and $\rho_0(1 - g)$ corresponds to $p_\pi(\pi)$. $\rho_N(1 - g)$ corresponds to $p(\pi|\nu = 0)$ with N examples in the analysis of chapter 2. In other words, the analysis of [Schwartz *et al.*, 1990] addresses the expected test error given zero training error, in our notation $E[\pi|\nu = 0]$.

In chapter 2 we present a mathematical paradigm for generalization which we call the *bin model* [Abu-Mostafa and Song, 1996]. The bin model relates a classification problem to a physical experiment that is easy to visualize, and which lends itself to probabilistic analysis. The most significant result gives the expected generalization error for a given level of training error (equation 2.10). Thus, when we formalize a learning process in terms of the bin model, we can directly find the quantity of interest – the expected performance on new data. Even in the cases where the quantity cannot be expressed in closed form, the equation still enables us to prove invariants of the learning process concerning generalization and overfitting without the need for a closed-form solution. The simplicity of the bin model enables us to accommodate noisy examples using a binary symmetric channel. Further analysis extends the model to regression problems as well as input-dependent noise.

1.3 Hardware Learning

In order to improve expected generalization, it is desirable to obtain as many examples as possible. With a very large data set, however, carrying out a learning algorithm may take a very long time. A major issue in learning from examples is the computation time required to carry out a learning algorithm. One direct way to improve the speed of learning is by moving from software to hardware.

For some time, there have been hardware implementations of learning systems, and in particular neural network models [Mead, 1989] [Ramacher and Ruckert, 1991]. More recently, a different approach to hardware machine learning has arisen. Instead of designing a hardware representation of an established learning system, a learning algorithm can be designed for use with existing reconfigurable hardware devices. Most of the results along these lines have involved genetic or evolutionary algorithms, and the field is generally known as *evolvable hardware* (EH). Some success has been shown in evolving analog and mixed-signal circuits [Thompson, 1998] [Bennett *et al.*, 1999], digital filters [Miller, 1999], control circuitry [Keymeulen *et al.*, 1998] and a variety of components for practical applications [Higuchi *et al.*, 1999]. Like Perkowski et al. [Perkowski *et al.*, 1999], we refer to this hardware adaptation as *learning hardware*, with EH as a special case.

In chapter 3 we look at applying learning techniques to a hardware learning model. We use a field programmable gate array (FPGA) which can be arbitrarily reprogrammed, allowing the flexibility of a parameterized learning model with function evaluations at hardware speeds. We show that digital circuits with low error rates can be learned from examples on the FPGA. Whereas similar approaches have primarily focussed on evolutionary algorithms, we show that use of an alternative optimization algorithm can result in improved performance.

Chapter 2 The Bin Model

In many learning problems, we deal with learning models and target functions that are nonlinear and data sets that contain some type of noise. These present theoretical difficulties, and thus much of the technical analysis in the field is highly specialized for a particular learning model, noise model, learning algorithm and problem class. In this chapter we present a framework for studying generalization that, for arbitrary target functions, arbitrary learning models, arbitrary input distributions and noisy data sets, can estimate the out-of-sample performance in terms of the in-sample performance, characterize the conditions for overfitting and quantify the effect of noise on generalization. We abstract the learning model, input distribution and target function for a given problem into a single univariate distribution which we will see suffices to completely describe the generalization behavior. We do, however, assume that the hypothesis selection is done according to a specific algorithm (although we discuss use of alternative algorithms in section 2.5).

2.1 Exhaustive Learning Algorithm

In many typical learning algorithms, a starting point is chosen, and a small set of hypotheses are explored according to some sequential rule. Gradient based methods [Rumelhart *et al.*, 1986], for example, typically explore only a few hypotheses that lie on a path of descent in parameter space. This results in a great speed advantage, but is susceptible to getting stuck in local minima. Global optimization techniques like simulated annealing [Kirkpatrick *et al.*, 1983] explore a much greater number of hypotheses. The error spaces corresponding to familiar parameterized learning models like artificial neural networks tend to have many symmetries [Bishop, 1995b], so there remains the question of selecting from hypotheses with the same in-sample performance.

In the bin model analysis, we consider hypotheses to be selected from the learning model in a way similar to that of the exhaustive learning paradigm discussed in section 1.2. Hence we refer to this as the *exhaustive learning algorithm.*

We assume there is a probability distribution $p_{\mathcal{G}}$ over the set of hypotheses in the learning model. We have some performance criterion that we wish to satisfy, and which is satisfied by a subset of the learning model, $\mathcal{G}^*$. The output of the learning algorithm is one hypothesis from $\mathcal{G}^*$ randomly selected according to the underlying distribution.

In some simple cases, we can implement exhaustive learning by finding $\mathcal{G}^*$ explicitly and making a random selection. It is often more practical, however, to repeatedly make random selections (with replacement) of g from $\mathcal{G}$, stopping when we find $g \in \mathcal{G}^*$. The two approaches are equivalent, with the selected hypothesis distributed as

$$p(\mathcal{A}(\mathcal{D}_N) = g) = \frac{p_{\mathcal{G}}(g)}{\int_{\mathcal{G}^*} p_{\mathcal{G}}(g)} \tag{2.1}$$

in either case.

2.2 The Bin Model for Classification

We introduce the bin model in the context of a binary classification problem, that is $f : \mathcal{X} \to \{0, 1\}$ and $g : \mathcal{X} \to \{0, 1\}$. We use the error function

$$\begin{aligned} e(g, x) &= (g(x) - f(x))^2 && (2.2) \\ &= \begin{cases} 0 & g(x) = f(x) \\ 1 & g(x) \neq f(x) \end{cases} && (2.3) \end{aligned}$$

We can envision a learning model as a collection of bins. Each bin ($g \in \mathcal{G}$) contains a number of marbles, each colored red or green. Each marble corresponds to a point $x \in \mathcal{X}$, and is colored green if $g(x) = f(x)$ and red otherwise. Then observing the behavior of a hypothesis g on a data set $\mathcal{D}$ corresponds to selecting a handful of marbles from the appropriate bin and observing their colors.

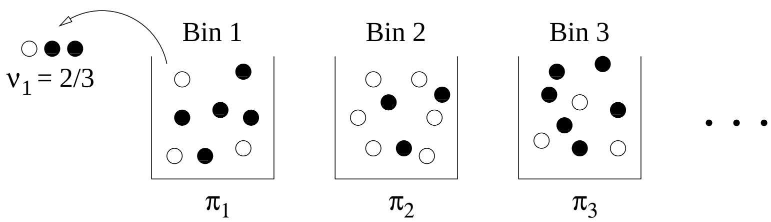


Figure 2.1: A set of bins containing colored marbles

Mathematically, green marbles correspond to correct classifications (0 error), and red marbles to incorrect classifications (1 error). Each hypothesis g has an inherent probability of error, which we denote by $\pi(g)$. $\pi(g)$ represents the fraction of red marbles in the corresponding bin.

$$\pi(g) = Pr[g(x) \neq f(x)] \tag{2.4}$$
$$= E_x[e(g,x)] \tag{2.5}$$

We denote the in-sample (training) error on the data set $\mathcal{D}$ by $\nu_{\mathcal{D}}$.

$$\nu_{\mathcal{D}}(g) = \frac{1}{N}\sum_{i=1}^{N} e(g,x) \tag{2.6}$$

The model is illustrated in figure 2.2. We have a set of bins, each with a corresponding value of π. A sample of three marbles selected from the first bin contains two red marbles. This corresponds to an in-sample error on a set of three points of 2/3 when we classify them with the first hypothesis.

For a given hypothesis with error probability π, the probability for an in-sample error ν is given by the binomial distribution,

$$P[\nu|\pi] = \binom{N}{N\nu}\pi^{N\nu}(1-\pi)^{N(1-\nu)} \tag{2.7}$$

We assume there is a probability distribution $p_{\mathcal{G}}$ on the set of hypotheses. $p_{\mathcal{G}}$ induces a π-distribution $p_\pi(\pi)$. We use the exhaustive learning algorithm of section 2.1 to select a hypothesis with in-sample error ν. We can then explicitly compute the expected out-of-sample error, $\pi(\nu) = E[\pi|\nu]$.

$$\pi(\nu) = \int_0^1 s\ p_{\pi|\nu}(s|\nu)ds \tag{2.8}$$

$$= \frac{\int_0^1 s\ p_{\pi,\nu}(s,\nu)ds}{\int_0^1 p_{\pi,\nu}(s,\nu)ds} \tag{2.9}$$

$$= \frac{\int_0^1 s\ p_{\pi}(s)s^{N\nu}(1-s)^{N(1-\nu)}ds}{\int_0^1 p_{\pi}(s)s^{N\nu}(1-s)^{N(1-\nu)}ds} \tag{2.10}$$

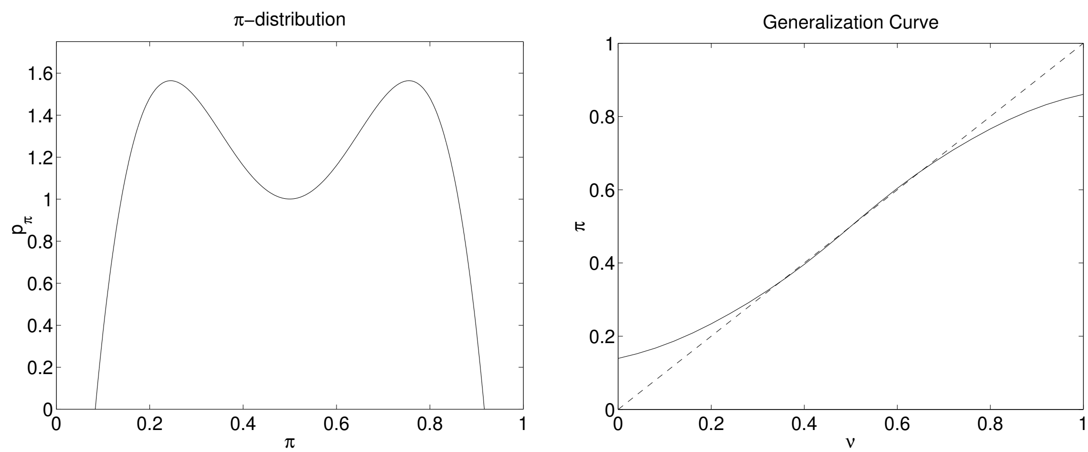
We refer to the relationship between $\pi(\nu)$ and ν in (2.10) as the *generalization curve*. A perfect generalization curve should be $\pi(\nu) = \nu$, which implies that in-sample error ν is a perfect indication of out-of-sample error in expectation. It is not the absolute value of ν and $\pi(\nu)$, but the deviation between them that characterizes the generalization behavior of a learning process. The further away a generalization curve is away from $\pi(\nu) = \nu$ at any point, the worse the generalization is at that point. It is important to point out that no assumption is made about the dependence among the hypotheses. The statistical dependence among the hypotheses in the learning model does not enter the π-distribution.

Figure 2.2 (b) is the generalization curve based on the π-distribution given in 2.2 (a). According to this model, when the training error ν is zero, the expected out-of-sample error is actually 0.17, indicating this sample error is an optimistic estimation of the corresponding out-of-sample error.

We say that a function $F(x)$ is monotonically increasing (decreasing) in x iff $x_1 \leq x_2 \Leftrightarrow F(x_1) \leq F(x_2)$ ($x_1 \leq x_2 \Leftrightarrow F(x_1) \geq F(x_2)$). The following theorem gives a qualitative relationship between in-sample and out-of-sample errors.

Theorem 2.2.1 *The expected test error $\pi(\nu)$ is monotonically increasing in the empirical error ν.*

See appendix A for the proof. Theorem 2.2.1 tells us that there is no overfitting with the exhaustive learning algorithm – to improve generalization, we should always find a hypothesis that does better on the training set.



(a) (b)

Figure 2.2: An example π-distribution and its generalization curve

For certain π-distributions, we can obtain $\pi(\nu)$ explicitly. For illustrative purposes we consider the case $p_\pi(\pi) = (d+1)\pi^d$, for $d \in \mathbf{Z}^+$ (nonnegative integers). The generalization curve is

$$\pi(\nu) = \frac{N\nu + d + 1}{N + d + 2}, \tag{2.11}$$

Of particular interest is the case $d = 0$. Then $p(\pi) = 1$ is the uniform distribution, which implies that we have no bias or preference over the distribution of π. $\pi(\nu)$ is linear in ν, but is skewed towards $\pi = \frac{1}{2}$ depending on the amount of data (see figure 2.3). If there is only one training example ($N = 1$), then $\pi(\nu) = \frac{1}{3}\nu + \frac{1}{3}$. As the number of examples grows ($N \to \infty$), $\pi(\nu) \to \nu$ indicating that the training error is a perfect indication of the generalization error, as is expected.

2.3 The Effect of Noise

2.3.1 Uniform Noise

In the case of a binary classification problem, we can consider noise in the output to be a random flip of the classification with some probability. We define the input-independent noisy version $\tilde{f}$ of the target function f as

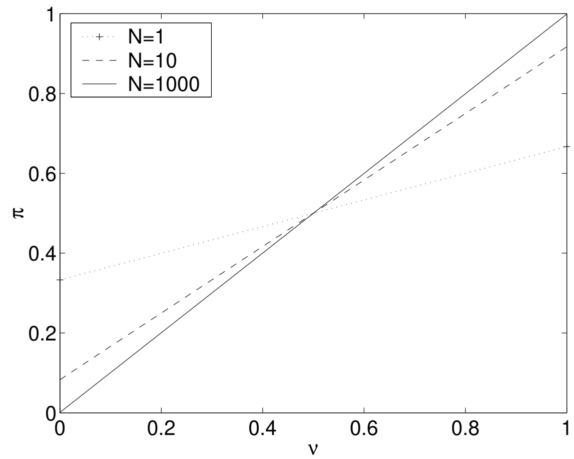


Figure 2.3: Generalization curves for a uniform π-distribution and varying training set size

$$\tilde{f}(x) = \begin{cases} f(x) & \text{with probability } 1 - \varepsilon_0 \\ 1 - f(x) & \text{with probability } \varepsilon_0 \end{cases} \tag{2.12}$$

This is the same type of noise present in a binary symmetric channel (BSC) [Cover and Thomas, 1991]. This BSC noise is easily incorporated into the Bin Model. Let $\tilde{\pi}(g)$ denote the error of a hypothesis g with the noisy target function $\tilde{f}$. We can write $\tilde{\pi}$ in terms of π.

$$\begin{aligned} \tilde{\pi}(g) &= Pr_x[g(x) \neq \tilde{f}(x)] && (2.13) \\ &= (1 - \varepsilon_0) Pr_x[g(x) \neq f(x)] + \varepsilon_0 Pr_x[g(x) = f(x)] && (2.14) \\ &= (1 - \varepsilon_0)\pi(g) + \varepsilon_0(1 - \pi(g)) && (2.15) \\ &= \pi(g)(1 - 2\varepsilon_0) + \varepsilon_0 && (2.16) \end{aligned}$$

If we know the distribution p_π, we can find the distribution of $\tilde{\pi}$ by a change of variables.

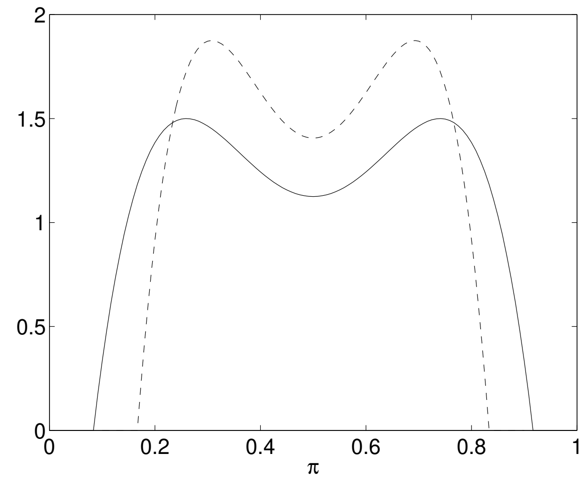


Figure 2.4: A π-distribution and the resulting $p_{\tilde{\pi}}$ under BSC noise

$$p_{\tilde{\pi}}(\tilde{\pi}) = \frac{p_{\pi}((\tilde{\pi} - \varepsilon_0)/(1 - 2\varepsilon_0))}{1 - 2\varepsilon_0} \tag{2.17}$$

Thus the result of adding BSC noise results in a linear transformation of π and a compression of the π-distribution. This effect is illustrated in figure 2.4. The π-distribution is squeezed towards $\pi = \frac{1}{2}$, indicating that the performance of every hypothesis is a bit closer to that of random guessing.

Now we can study $\tilde{\pi}(\nu) = E[\tilde{\pi}|\nu]$. Theorem 2.2.1 immediately applies – $\tilde{\pi}(\nu)$ is monotonically increasing in ν, thus for $\varepsilon_0 < \frac{1}{2}$, $\pi(\nu)$ is also monotonic. Thus there is no overfitting, even in the case of a noisy data set.

2.3.2 Input Dependent Noise

The noise in a data set may be input dependent. For example, on an experimental measurement, the uncertainty can depend on the characteristics of the particular instrument used, and may change for different measurement scales.

We model this by writing the noise $\varepsilon = \varepsilon(x)$. We are now concerned with the probability of the hypothesis agreeing with the *noisy* target value. Once again we define a noisy target function $\tilde{f}$.

$$\tilde{f}(x) = \begin{cases} f(x) & \text{with probability } 1-\varepsilon(x) \\ 1-f(x) & \text{with probability } \varepsilon(x) \end{cases} \tag{2.18}$$

$$\begin{aligned} P[g(x) \neq \tilde{f}(x)|x] &= e(g,x)(1-\varepsilon(x)) + (1-e(g,x))\varepsilon(x) && (2.19) \\ \tilde{\pi}(g) &= P[g(x) \neq \tilde{f}(x)] && (2.20) \\ &= E_x[e(g,x)(1-\varepsilon(x)) + (1-e(g,x))\varepsilon(x)] && (2.21) \\ &= E_x[e(g,x)] + E_x[\varepsilon(x)] - 2E_x[e(g,x)\varepsilon(x)] && (2.22) \\ &= \pi(g) + E_x[\varepsilon(x)] - 2E_x[e(g,x)\varepsilon(x)] && (2.23) \end{aligned}$$

Of particular interest are noise models which allow $\tilde{\pi}(g)$ to be determined given only $\pi(g)$.

Definition 2.3.1 *We say a noise model $\varepsilon(x)$ is* **regular** *iff $\tilde{\pi}(g) = \tilde{\pi}(\pi(g))$.*[1]

In the case of constant noise $\varepsilon(x) = \varepsilon_0 \; \forall x$, (2.23) reduces to the result of section 2.3.1, $\tilde{\pi} = \pi(1-2\varepsilon_0) + \varepsilon_0$.

We illustrate the case of input dependent noise with a simple example. Consider the learning model that consists of simple threshold functions, $g_w(x) = sgn(x - w), w \in [0,1]$. Furthermore, assume that the target function is in the learning model, $f(x) = g_\alpha(x)$, and for simplicity that $\frac{1}{2} < \alpha < 1$. Assume that the input distribution, is uniform in $(0,1)$, and that exhaustive learning is used with w chosen uniformly in $(0,1)$.

For this simple learning problem, an error occurs ($e(g_w, x) = 1$) when either $w < x < \alpha$ or $\alpha < x < w$. It follows that $\pi(g_w) = |\alpha - w|$. Also for any noise function $\varepsilon(x)$ we can compute $\tilde{\pi}(g_w)$.

$$\tilde{\pi}(g_w) = \begin{cases} \pi(g_w) + \int_0^1 \varepsilon(x)dx - 2\int_w^\alpha \varepsilon(x)dx & w < \alpha \\ \pi(g_w) + \int_0^1 \varepsilon(x)dx - 2\int_\alpha^w \varepsilon(x)dx & w > \alpha \end{cases} \tag{2.24}$$

[1]Equivalently, we need only say that $E_x[e(g,x)\varepsilon(x)]$ is a function of $\pi(g)$.

We notice that for $p < (1-\alpha)$ there are two hypotheses, $g_{\alpha-p}$ and $g_{\alpha+p}$ that have $\pi(g) = p$. For $(1-\alpha) < p < \alpha$ there is only one such hypothesis, and none has $\pi > \alpha$. Thus the noise model is regular if it satisfies $\tilde{\pi}(g_{\alpha-p}) = \tilde{\pi}(g_{\alpha+p})$ for all $p < (1-\alpha)$. For regular noise distributions, we can rewrite the noisy error $\tilde{\pi}$ as a function of the noiseless error π.

$$\tilde{\pi}(\pi) = \tilde{\pi}(g_{\alpha-\pi}) \tag{2.25}$$

$$= \pi(g_w) + \int_0^1 \varepsilon(x)dx - 2\int_{\alpha-\pi}^{\alpha} \varepsilon(x)dx \tag{2.26}$$

Given a training error ν on N points, we are interested in computing the noisy and noiseless expected test errors, $\tilde{\pi}(\nu)$ and $\pi(\nu)$ respectively.

$$P_{\nu|g}[\nu|g] = P_{\nu|\tilde{\pi}}[\nu|\tilde{\pi}(g)] \tag{2.27}$$

$$= \binom{N}{N\nu} \tilde{\pi}(g)^{N\nu}(1-\tilde{\pi}(g))^{N(1-\nu)} \tag{2.28}$$

$$\tilde{\pi}(\nu) = E_{g,x^N}[\tilde{\pi}|\nu] \tag{2.29}$$

$$= \int_0^1 s\ p_{\tilde{\pi}|\nu}[s|\nu]ds \tag{2.30}$$

$$= \frac{\int_0^1 s\ P_{\nu|\tilde{\pi}}[\nu|s]p_{\tilde{\pi}}(s)ds}{\int_0^1 P_{\nu|\tilde{\pi}}[\nu|s]p_{\tilde{\pi}}(s)ds} \tag{2.31}$$

$p_{\tilde{\pi}}(\cdot)$ is the distribution of $\tilde{\pi}(g)$ induced by $p_{\mathcal{G}}(g)$. When the noise distribution is regular, we can write $\tilde{\pi}(\nu)$ and $\pi(\nu)$ in terms of the π-distribution, $p_{\pi}(\cdot)$.

$$P_{\nu|\pi}[\nu|\pi] = P_{\nu|\tilde{\pi}}[\nu|\tilde{\pi}(\pi)] \tag{2.32}$$

$$= \binom{N}{N\nu} \tilde{\pi}(\pi)^{N\nu}(1-\tilde{\pi}(\pi))^{N(1-\nu)} \tag{2.33}$$

$$\pi(\nu) = E_{g,x^N}[\pi|\nu] \tag{2.34}$$

$$= \int_0^1 s\ p_{\pi|\nu}[s|\nu]ds \tag{2.35}$$

$$= \frac{\int_0^1 s \; P_{\nu|\pi}[\nu|s] p_\pi(s) ds}{\int_0^1 P_{\nu|\pi}[\nu|s] p_\pi(s) ds} \tag{2.36}$$

$$\tilde{\pi}(\nu) = \frac{\int_0^1 \tilde{\pi}(s) P_{\nu|\pi}[\nu|s] p_\pi(s) ds}{\int_0^1 P_{\nu|\pi}[\nu|s] p_\pi(s) ds} \tag{2.37}$$

In general, computing exact values for $\pi(\nu)$ and $\tilde{\pi}(\nu)$ is intractable. However, some general overfitting results can be obtained, namely, noisy test error is a monotonically increasing function of the noisy training error. Furthermore, when $\tilde{\pi}$ is a monotonically increasing function of π, the noiseless test error is also a monotonically increasing function of the *noisy* training error.

Theorem 2.3.1 *For any $\tilde{\pi}(g)$, $p_{\mathcal{G}}(\cdot)$ and noise $\varepsilon(x)$, $\tilde{\pi}(\nu)$ is monotonically increasing in ν.*

Theorem 2.3.2 *If $\varepsilon(x)$ is regular, then the following are equivalent:*

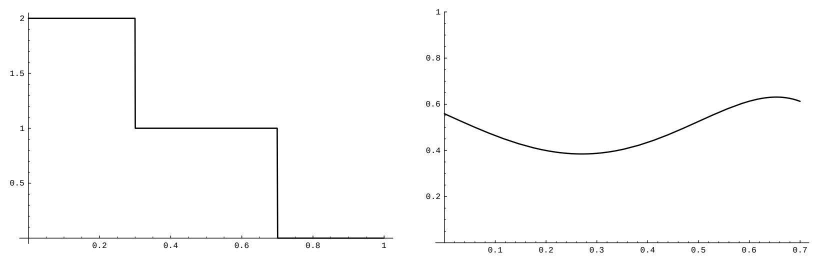
- *$\tilde{\pi}(\pi)$ is monotonically increasing (decreasing) in π*
- *for any $p_\pi(\cdot)$, $\pi(\nu)$ is monotonically increasing (decreasing) in ν*

Theorem 2.3.1 tells us that overfitting will never be observed in the noisy out-of-sample error, and is a generalization of Theorem 2.2.1. Theorem 2.3.2 gives conditions on the noise under which overfitting will or will not occur in the noiseless out-of-sample error. For proofs see appendix A.

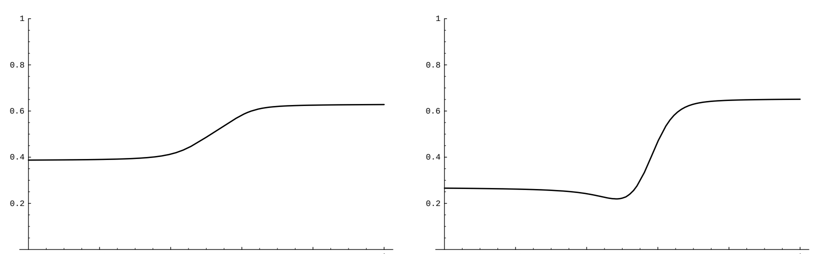
Figure 2.5 shows several interesting quantities for the example above with $\alpha = 0.7$ and $\varepsilon(x) = 16(x-\alpha)^4 - 8(x-\alpha)^2 + 1$. This choice of $\varepsilon(x)$ is regular for this learning problem, and we see that $\tilde{\pi}(\pi)$ is not monotonically increasing in π, and for the given $p(\pi)$, $\pi(\nu)$ is not monotonically increasing in ν.

2.3.3 Value of Noisy Examples

In [Magdon-Ismail *et al.*, 1998], Magdon-Ismail et al. studied how noise affects generalization for a general class of learning systems. They showed that the generalization error for a learning problem with noise can be bounded by a function of the generalization error with noiseless data. Explicitly, they found that



(a) $p(\pi)$ (b) $\tilde{\pi}(\pi)$



(c) $\tilde{\pi}(\nu)$ (d) $\pi(\nu)$

Figure 2.5: π-distribution, noise dependence and generalization curves for an input-dependent noise example

$$\mathcal{E}_N(\sigma) \leq \mathcal{E}_N(0) + \frac{C_1\sigma^2 + C_2}{N} + o(\frac{1}{N}) \tag{2.38}$$

where $\mathcal{E}_N(\sigma)$ is the expected test error with N training examples and noise variance σ^2, and C_1 and C_2 are constants depending on the learning problem.[2]

We can derive a similar result for the bin model with exhaustive learning. For simplicity we consider the case that $\nu = 0$ and a uniform π-distribution, $p_\pi(\pi) = 1, \pi \in [0, 1]$. We assume the noise is of the BSC type with probability of flip ε_0.

$$\begin{aligned}
E[\pi|\nu = 0] &= \frac{\int_0^1 \pi(1-\pi)^N p_\pi(\pi) d\pi}{\int_0^1 (1-\pi)^N p_\pi(\pi) d\pi} && (2.39)\\
&= \frac{1}{N+2} && (2.40)\\
E[\pi|\nu = 0, \varepsilon_0] &= \frac{\int_0^1 \pi(1-\tilde{\pi})^N p_\pi(\pi) d\pi}{\int_0^1 (1-\tilde{\pi})^N p_\pi(\pi) d\pi} && (2.41)\\
&= \frac{\int_0^1 \pi(1-(1-2\varepsilon_0)\pi - \varepsilon_0)^N d\pi}{\int_0^1 (1-(1-2\varepsilon_0)\pi - \varepsilon_0)^N d\pi} && (2.42)\\
&= \frac{1}{N+2}\left(1 + \frac{\varepsilon_0}{1-2\varepsilon_0} - \frac{\varepsilon_0^{N+1}}{\varepsilon_0^{N+1} - (1-\varepsilon_0)^{N+1}}\right) \\
&\quad + \frac{\varepsilon_0^{N+1}}{\varepsilon_0^{N+1} - (1-\varepsilon_0)^{N+1}} && (2.43)\\
&= E[\pi|\nu = 0] + \frac{1}{N+2}\left(\frac{\varepsilon_0}{1-2\varepsilon_0}\right) + \Theta(e^{-\alpha N}) && (2.44)
\end{aligned}$$

for some $\alpha > 0$, where we have assumed that $\varepsilon_0 < \frac{1}{2}$ for convenience.[3]

This result tells us what (noiseless) generalization error we can expect using exhaustive learning if we can estimate the noise in the training data and we are guaranteed to find $\nu = 0$. The explicit result is only for $p_\pi(\pi) = 1$, but similar results can be derived for other π-distributions.

To compare this result with (2.38), we need to write (2.44) in terms of σ^2. Since $\sigma^2 = \varepsilon_0(1 - \varepsilon_0) \in [0, \frac{1}{4})$ for the BSC noise model, we can write

[2] We write $g(x) = o(f(x))$ iff $\lim_{x\to\infty} g(x)/f(x) = 0$.

[3] We write $g(x) = \Theta(f(x))$ iff $\lim_{x\to\infty} g(x)/f(x) \in (0, \infty)$.

$$E[\pi|\nu = 0, \varepsilon_0] \;=\; E[\pi|\nu = 0] + \frac{1}{N+2}\left(\frac{1}{2\sqrt{1-4\sigma^2}} - 1\right) + \Theta(e^{-\alpha N}) \quad (2.45)$$

We see that the number of training examples plays a similar role in (2.38) and (2.45), but that the effect of noise is somewhat different.

2.4 The Bin Model for Regression

So far, the bin model analysis has dealt only with classification problems. In order to incorporate continuous error measures and regression problems, we need to generalize the concept of π. We now redefine π in terms of an arbitrary error function $e : \mathcal{G}\times\mathcal{X} \rightarrow \mathcal{E}$ where $\mathcal{E} = [e_{min}, e_{max}]$, $e_{min} = inf(e)$ and $e_{max} = sup(e)$.

For any hypothesis g, the error values $e(g, x)$ are distributed according to some distribution $p_{e|g}(e|g)$. Again we denote the expected generalization error by $\pi(g)$.

$$\begin{aligned}
\pi(g) &= E_x[e(g,x)] && (2.46)\\
&= \int_{\mathcal{X}} e(g,x)p_{\mathcal{X}}(x)dx && (2.47)\\
&= \int_{e_{min}}^{e_{max}} s\ p_{e|g}(s|g)ds && (2.48)
\end{aligned}$$

where $p_{e|g}$ is the distribution induced on e by $p_{\mathcal{G}}$ and $p_{\mathcal{X}}$.

For the regression analysis, we will also need to find the variance of the errors for a given hypothesis, which we denote by $\sigma^2(g)$.

$$\begin{aligned}
\sigma^2(g) &= E_x[(e(g,x) - \pi(g))^2] && (2.49)\\
&= \int_{e_{min}}^{e_{max}} (s - \pi(g))^2 p_{e|g}(s|g)ds && (2.50)
\end{aligned}$$

For a general error function, π and ν now can take any values in $\mathcal{E}$. Once again, the quantity of interest is the expected generalization error $\pi(\nu)$. In the binary classification problem, we found that $\pi(\nu)$ was a monotonically increasing function of ν. For regression problems, this is not necessarily true. We would like to determine the conditions under which $\pi(\nu)$ is always monotonically increasing in ν, that is, under which $\frac{d}{d\nu}\pi(\nu) \geq 0$ for all ν.

2.4.1 Monotonicity Conditions

As with classification problems, we would like to abstract the characteristics of the learning problem into the π-distribution. Given p_π, we can evaluate $\pi(\nu)$ if we assume that, given $\pi(g)$, there is some conditional distribution $p_{\nu|\pi}$ for the in-sample error. We can determine conditions on $p_{\nu|\pi}$ that guarantee monotonicity of the generalization curve.

Theorem 2.4.1 *If $p_{\nu|\pi}$ is non-zero, continuous and differentiable on $(\nu, \pi) \in \mathcal{E}^2$, then $\pi(\nu)$ is monotonically increasing in ν for any p_π iff $(\frac{\partial}{\partial\nu}p_{\nu|\pi})/p_{\nu|\pi}$ is monotonically increasing in π.*

See appendix A for the proof. Thus, if we can specify how the in-sample errors arise as a function of the out-of-sample error, we may be able to rule out overfitting under exhaustive learning. In general, however, specifying such a $p_{\nu|\pi}$ is very difficult. Something can be said, though, in the large data limit.

The in-sample error is

$$\nu = \frac{1}{N}\sum_{i=1}^{N} e(g, x_i) \in \mathcal{E} \tag{2.51}$$

Since each x_i is i.i.d. according to the input distribution, each $e(g, x_i)$ is i.i.d. according to some distribution $p_{e|g}$. Usually we don't know $p_{e|g}$, but we do know that it has mean $\pi(g)$ and variance $\sigma^2(g)$. The central limit theorem tells us that as $N \to \infty$,

$$\nu \rightsquigarrow \mathcal{N}(\pi(g), \frac{\sigma^2(g)}{N}) \tag{2.52}$$

That is, $p_{\nu|g}$ converges in distribution to a Gaussian with mean $\pi(g)$ and variance $\sigma^2(g)/N$.

$$p_{\nu|\pi,\sigma}(\nu|\pi,\sigma) \propto \exp(-\frac{N(\nu-\pi(g))^2}{2\sigma^2(g)}) \tag{2.53}$$

If for any g, the error variance is a function of the mean,[4] that is, $\sigma(g) = \sigma(\pi(g))$, then we can rewrite (2.53) and apply Theorem 2.4.1 to find necessary and sufficient conditions for monotonicity of $\pi(\nu)$.

$$\begin{aligned} p_{\nu|\pi}(\nu|\pi) &\propto \exp(-\frac{N(\nu-\pi)^2}{2\sigma^2(\pi)}) && (2.54) \\ \frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi) &= \frac{N(\pi-\nu)}{\sigma^2(\pi)}p_{\nu|\pi}(\nu|\pi) && (2.55) \end{aligned}$$

Theorem 2.4.2 *If $\sigma(g) = \sigma(\pi(g))$ $\forall g \in \mathcal{G}$, then in the large data limit $N \to \infty$, $\pi(\nu)$ is monotonically increasing in ν for all p_π iff $\frac{e_{max}-\pi_1}{e_{max}-\pi_2} \leq \frac{\sigma^2(\pi_1)}{\sigma^2(\pi_2)} \leq \frac{\pi_1-e_{min}}{\pi_2-e_{min}}$ whenever $\pi_1 > \pi_2$.*

Theorem 2.4.2 gives a condition on the learning problem such that under exhaustive learning, asymptotically there will be no overfitting. The form of $\sigma(\pi)$ will generally depend on the learning model and target function. We can find common distributions for which the condition either satisfied or violated. For the Bernoulli distribution (with $\mathcal{E} = \{0,1\}$), $\sigma^2(\pi) = \pi(1-\pi)$ and the condition holds, reiterating the results of section 2.2. For a uniform distribution over $\mathcal{E} = \{0, e_{max}(g)\}$, however, $\pi(g) = e_{max}(g)/2$ and $\sigma^2(\pi) = \pi^2/3$, so the condition is violated.

2.4.2 Noise in Regression Problems

For the purposes of analyzing the effects of noise in the case of regression problems, it is useful to work in the frequency domain. Letting $\hat{p}$ denote the Fourier transform

[4] This is true for many familiar distributions, for example, when $p_{e|g}$ is binomial, exponential or χ^2.

of p, we have

$$
\begin{aligned}
p_{\nu|g}(\nu|g) &= \prod^{\star} p_{e|g}(e(g,x_i)|g) && (2.56)\\
\widehat{p_{\nu|g}}(\omega) &= \widehat{p_{e|g}}^{N}(\omega) && (2.57)
\end{aligned}
$$

That is, $p_{\nu|g}$ is the N-fold convolution of $p_{e|g}$ with itself. We are interested in quantifying the effects of noise on $p_{e|g}$. This allows us to find $p_{\nu|g}$ for any $g \in \mathcal{G}$, allowing the computation of $p_{\nu|\pi}$ and hence the expected out-of-sample error $\pi(\nu)$. We consider the familiar case of a squared error measure and additive zero-mean Gaussian noise.

$$
\begin{aligned}
e(g,x) &= (g(x) - f(x))^2 && (2.58)\\
\tilde{f}(x) &= f(x) + \eta && (2.59)\\
\eta &\sim \mathcal{N}(0, \sigma_\eta^2) && (2.60)\\
\tilde{e}(g,x) &= (g(x) - \tilde{f}(x))^2 && (2.61)
\end{aligned}
$$

In this case we can exactly determine the relationship in the frequency domain between the noiseless error distribution $p_{e|g}$ and the noisy version $p_{\tilde{e}|g}$.

$$
\widehat{p_{\tilde{e}|g}}(\omega) = \frac{\widehat{p_{e|g}}\left(\frac{\omega}{1+2i\sigma_\eta^2\omega}\right)}{\sqrt{1+2i\sigma_\eta^2\omega}} \qquad (2.62)
$$

A derivation is given in Proposition A.0.1 in the appendix. Equation (2.62) gives us a transformation that is useful if the entire distribution $p_{e|g}$ is known. In the noiseless case, however, we could guarantee monotonicity of $\pi(\nu)$ knowing only $\pi(g)$ and $\sigma^2(g)$. With a squared error measure it is straightforward to determine these parameters for the noisy case in terms of the noiseless versions and the moments of the noise distribution p_η. For zero-mean, zero-skew input-independent noise

$$\tilde{\pi}(g) = E_{x,\eta}[(g(x) - \tilde{f}(x))^2] \quad (2.63)$$

$$= E_x[(g(x) - f(x))^2] - 2E_x[(g(x) - f(x))]E_\eta[\eta] + E_\eta[\eta^2] \quad (2.64)$$

$$= \pi(g) + \sigma_\eta^2 \quad (2.65)$$

$$\tilde{\sigma}^2(g) = E_{x,\eta}[(g(x) - \tilde{f}(x))^4] - \tilde{\pi}(g)^2 \quad (2.66)$$

$$= E_x[(g(x) - f(x))^4] + 4E_x[(g(x) - f(x))^3]E_\eta[\eta]$$

$$+6E_x[(g(x) - f(x))^2]E_\eta[\eta^2] + 4E_x[g(x) - f(x)]E_\eta[\eta^3]$$

$$+E_\eta[\eta^4] - (\pi(g) + \sigma_\eta^2)^2 \quad (2.67)$$

$$= \sigma^2(g) + 4\pi(g)\sigma_\eta^2 + E_\eta[\eta^4] - \sigma_\eta^4 \quad (2.68)$$

In the special case of Gaussian noise, we can simplify (2.68) to get an expression for $\tilde{\sigma}^2(g)$ that depends only on $\pi(g)$, $\sigma^2(g)$ and the noise variance σ_η^2.

$$\tilde{\sigma}^2(g) = \sigma^2(g) + 4\pi(g)\sigma_\eta^2 + 2\sigma_\eta^4 \quad (2.69)$$

Hence, as in section 2.3.1, the addition of input-independent noise results in a simple transformation of the essential parameters $\pi(g)$ and $\sigma^2(g)$. We can then apply Theorem 2.4.2 with the noisy $\tilde{\pi}(g)$ and $\tilde{\sigma}^2(g)$ to determine whether the monotonicity conditions are satisfied. For example, we consider the case of a Bernoulli distribution for $e(g, x) \in \{0, 1\}$ so that $\sigma^2 = \pi(1 - \pi)$. If we assume gaussian noise and use the transformations above, we have monotonicity of $\tilde{\pi}(\nu)$ when the following inequalities are true for all $\pi_1 > \pi_2$.

$$1 \leq \frac{\pi_1(1 - \pi_1) + 4\pi_1\sigma_\eta^2 + 2\sigma_\eta^4}{\pi_2(1 - \pi_2) + 4\pi_2\sigma_\eta^2 + 2\sigma_\eta^4} \leq \frac{\pi_1 + \sigma_\eta^2}{\pi_2 + \sigma_\eta^2} \quad (2.70)$$

The left inequality holds if $\sigma_\eta^2 > 1/4$, but if $p_\pi(\pi) > 0$ in some neighborhood of 0, then the right inequality is only satisfied for $\sigma_\eta^2 = 0$. In other words, the addition of Gaussian noise makes overfitting a possibility for this problem (which has monotonic

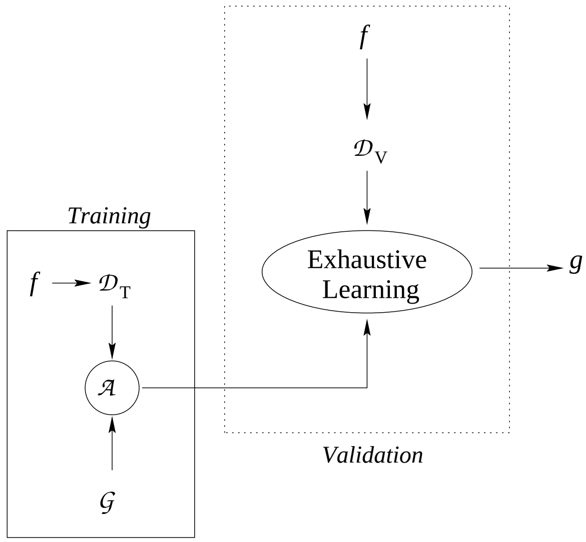


Figure 2.6: Incorporating a learning algorithm into the bin model framework

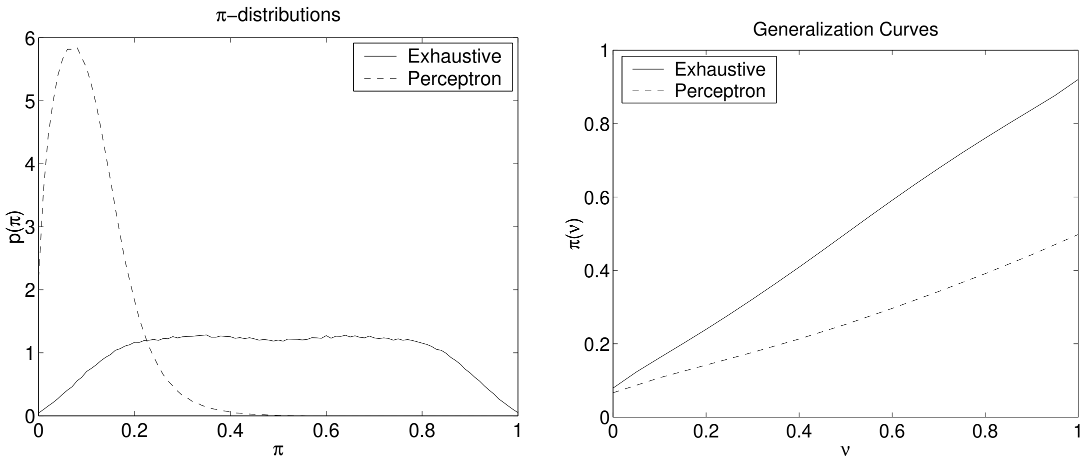
generalization in the noiseless case).

2.5 Other Learning Algorithms

One of the major limitations of the bin model thus far is that it requires an exhaustive learning algorithm. In real applications, the exhaustive learning algorithm is impractical. Results similar to Theorem 2.2.1 can be obtained in special cases [Cataltepe *et al.*, 1999], but in general the selected hypotheses will depend on the training set in ways specific to the individual learning algorithm. Nevertheless, there is a way that we can use an arbitrary learning algorithm and still take advantage of the bin model analysis through use of a validation set[Stone, 1974].

Given a data set $\mathcal{D}$, we divide it into a training set $\mathcal{D}_T$ and a validation set $\mathcal{D}_V$. The training set will be used to select a hypothesis using a learning algorithm $\mathcal{A}$. We have altered the black-box that produces the hypotheses. $p_{\mathcal{G}}(g)$ is now the distribution induced by $\mathcal{A}$ on the distribution of possible training sets. The new process is illustrated in figure 2.6.

There is now a training set underlying the selection process, but for the purpose of generalization analysis, the learning problem is the same. The hypotheses are



(a) (b)

Figure 2.7: Empirical generalization results for exhaustive and perceptron learning

produced according to some distribution $p_{\mathcal{G}}$, and we can find the generalization curve with respect to p_π, this time using $\mathcal{D}_V$ to compute the in-sample error. We now have a smaller number of examples, but presumably the hypotheses produced are better.

As an example of the improvement we can expect by using this two step learning, we consider a two dimensional classification problem. The target function f is a linear classifier, and $\mathcal{G}$ is a linear perceptron learning model. Figure 2.7 compares the π-distributions and generalization curves for exhaustive learning and the perceptron learning rule. While the π-distribution for the original $p_{\mathcal{G}}$ is relatively flat, using 10 examples to train with the perceptron skews the distribution to favor hypotheses with $\pi < 0.2$ (figure 2.7(a)). As a result, if we have 20 data points, then for any observed ν (on the validation set) we expect a much lower generalization error using the perceptron learning algorithm (figure 2.7(b)).

Although in general we might expect better generalization with more data, we can sacrifice some of it to incorporate a learning algorithm that alters the π-distribution. This allows us to overcome the restriction to the inefficient exhaustive learning algorithm.

2.6 Conclusion

We have presented a theoretical framework for studying the generalization behavior of learning systems. The bin model analysis parameterizes the generalization curve for a learning process in terms of its π-distribution. We have shown how this can be done for classification problems and certain classes of regression problems. The effects of noise on learning and generalization have been discussed in the context of the bin model. A method for incorporating the analysis with a given learning algorithm has been presented, and it was shown how this can lead to improved generalization performance.

The bin model is intended to be applicable to a very general learning problem, while addressing important issues and providing useful insights into the problem of generalization. While in general the π-distribution is unknown, certain invariants hold true. Using the exhaustive learning algorithm, overfitting is certain not to occur, even in the presence of regular noise. The same holds true for other algorithms in the validation scenario of section 2.5.

Chapter 3 Learning in Hardware

In general it is desirable to have a learning system learn as quickly as possible, and, once a hypothesis has been chosen, to produce outputs quickly when presented with inputs. There are some problems for which speed is critical. Real time audio and video processing algorithms may be sufficiently complex that a software implementation is too slow. For the most part these algorithms can be implemented with a digital signal processor or dedicated hardware, but for pattern recognition problems (for speech recognition or computer vision for example) the available algorithms may not lend themselves to such an implementation.

Furthermore, the process of designing and implementing an application specific integrated circuit (ASIC) is typically a long and expensive one. Field programmable gate arrays (FPGAs) are available as an alternative to reduce the concept-to-product time and the cost of making modifications. Recent FPGAs have computational resources on the order of hundreds of thousands or millions of gate equivalents, can be reprogrammed indefinitely and have in-circuit and partial reconfiguration possibilities.

FPGAs are very well suited for evolvable hardware, because of their low cost, rapid reconfigurability and near ASIC speed. FPGAs have been used as a platform for accelerating EH [Koza *et al.*, 1998], and also as a raw parameterized learning model [Thompson, 1998]. In the second case, evolutionary techniques have made use of unconventional properties of the physical device, yielding designs that defy conventional analysis [Thompson and Layzell, 1999].

It is this unrestricted model that interests us. Part of the advantage of EH is the removal of conventional digital design constraints. We do not wish to arbitrarily add new ones by imposing our own structure on the hardware device. Unfortunately this presents problems for a genetic representation of the model. Without some such structure, genetic operators have little meaning. We are therefore interested in other

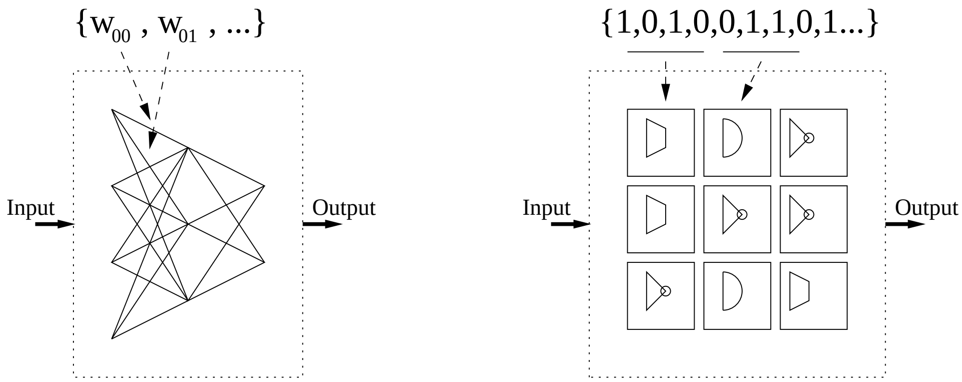


Figure 3.1: Comparison of a neural network with the hardware learning model

optimization techniques for maximizing a fitness criterion.

3.1 Learning Hardware

In the machine learning paradigm, we look at the configurable hardware as a parameterized learning model. The configuring bit string is the vector of (binary) parameters. We have some error (or fitness) criterion that we can evaluate for any hypothesis in the learning model (i.e. any configuration of the hardware device). We illustrate this in figure 3.1 by giving a comparison with a neural network learning model. The neural network is parameterized in terms of its weights, the hardware in terms of its configuring bit string. To a learning algorithm, each is simply a 'black box' that produces an output value for a given input.

In this model, the problem of hardware design is simply one of discrete optimization. We refer to the problem as one of error minimization rather than fitness maximization, although the two are equivalent. For this minimization we consider two alternative techniques, the genetic algorithm [Goldberg, 1989] and the reactive tabu search [Battiti and Tecchiolli, 1994].

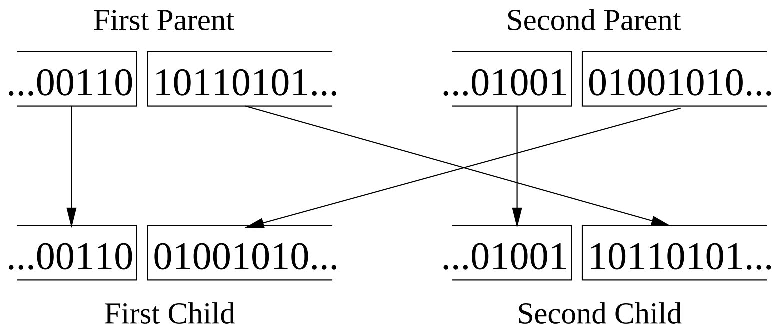


Figure 3.2: The genetic crossover operation

3.1.1 Genetic Algorithm

The genetic algorithm (GA) is meant to mimic Darwinian evolution. A population of candidates is maintained, and goes through a series of generations. For each new generation, some of the existing candidates survive, while others are created by a type of reproduction from a set of 'parents.'

The configuring bits $\vec{g} = (g_1, g_2, \ldots, g_L) \in \{0,1\}^L$ act as the genome for each candidate, and genetic variation is introduced by means of the genetic operators of *mutation* and *crossover*. In the GA used here, crossover is effected by splitting the genomes of two parents at some point and splicing the mismatched pieces as illustrated in figure 3.1.1. Mutations μ_i are pointwise (as in equation 3.1), with single bits in the genome being flipped with some probability p_{mut}.

$$\mu_i(\vec{g}) = (g_1, \ldots, g_{i-1}, 1 - g_i, g_{i+1}, \ldots, g_L) \tag{3.1}$$

All members of the population are subject to mutations from one generation to the next.

For the experiments of section 3.3, we have the best one third of the population survive and become potential parents. Parents are selected randomly with a probability based on rank. p_{mut} is chosen so that in any generation 2 mutations are expected in each genome.

3.1.2 Reactive Tabu Search

The reactive tabu search (RTS) is a hill-climbing algorithm that includes a *tabu* parameter that prevents undoing recently taken steps. This is intended to allow escape from local minima. In order to prevent cycles, the tabu parameter is adaptive.

For the experiments described here we have followed closely the implementation of RTS in [Battiti and Tecchiolli, 1995]. For a configuration $\vec{g}$, all changes are single bit flips $\mu_i(\vec{g})$ (as in equation 3.1). At each iteration, some subset $\mathcal{S}$ of possible changes to the configuration is considered, and the best change μ is found. The move μ is then 'tabu' for some time T, that is, it cannot be included in the subsequent $\mathcal{S}$.

When we encounter short cycles, we increase the tabu period $T \leftarrow \alpha T, \alpha > 1$. Thus, if we are in a local minimum and repeatedly undo short upward steps, T will increase until the tabu period is long enough to allow escape. When we encounter new configurations or very long cycles, we decrease the tabu period $T \leftarrow \beta T, \beta < 1$. In this way we allow refinement to a new minimum after escaping from another basin.

For the experiments of section 3.3, we take $|\mathcal{S}| = L/16$ (that is $1/16$ of all allowable mutations are considered), and use tabu adjustment parameters $\alpha = 1.1$ and $\beta = 0.9$.

3.2 Experimental Setup

3.2.1 The XC6216

The Xilinx XC6216 FPGA is particularly desirable for EH due to its partial reconfigurability, multiplexor based architecture and open architecture [Xilinx, Inc., 1997]. Partial reconfiguration allows the incremental changes of a learning algorithm to be made in hardware very rapidly. The MUX based architecture ensures that signal contention is not a problem, and thus that arbitrary configuration data will not cause the chip to self destruct.

The XC6216 has 4096 cells, each of which has a function unit with a register and nearest neighbor routing resources. Longer connections and special connections for arithmetic functions are available, but for regularity we have not used them. We

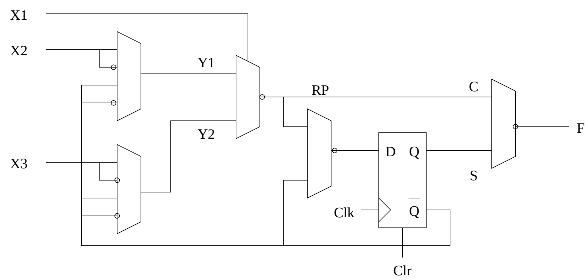


Figure 3.3: XC6216 function unit

also disable the register so that the circuit is purely combinational. This results in a configuration space that has 18-bits for each cell. The function unit in each cell consists of 5 multiplexors and a register and is illustrated in figure 3.2.1. It has three inputs and can implement any constant, one- or two-input input logic function or 2-to-1 MUX.

3.2.2 System Implementation

The hardware learning task was implemented intrinsically, that is, with the learning taking place on actual hardware. We use the Virtual Computer Corp. H.O.T.Works system [Schewel, 1997], which provides a PCI interface to the XC6216 chip and a C++ API. The board is installed in a Pentium Pro 200 PC. While the search algorithm is executed in software, each function evaluation is done by the FPGA device.

One quarter of the FPGA is reserved for experimentation (dynamic reconfiguration), a memory interface and control circuitry are placed on the periphery, and the remaining resources are available to facilitate wiring. A schematic of the static layout is given in figure 3.2.2.

The memory is clocked at 16MHz, and the examples are cycled at 1/16 the memory clock speed to allow settling of unstable or or oscillatory circuits.[1] For fitness evaluation, a set of input patterns is written to the on-board RAM. Each is presented

[1]We make no effort to disallow unstable circuits, but instead view the outputs as possibly noisy versions of the desired signal. Thus a signal that oscillates between logic 1 and logic 0 may, averaged over time be taken as having a value of 0.75. In the world of digital circuit design, this is undesirable, but in the realm of learning we may be dealing with an ill defined problem for which even human experts cannot determine the correct output with certainty.

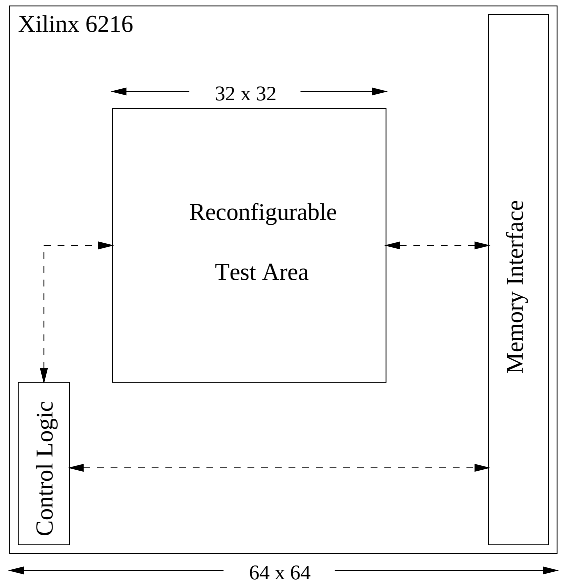


Figure 3.4: Schematic FPGA layout

for $1\mu s$, and the result is stored on-board. After all patterns have been presented, the results are read back to system memory, and compared to the targets in software. As we discuss in the next section, each iteration of a learning algorithm takes 10s to 100s of milliseconds. At $1\mu s$ per example presentation, the actual function evaluation accounts for a very small fraction ($< 1\%$) of the total execution time. This is a contrast to traditional learning models (e.g. neural networks), where computation of the function value may be complicated and may take up most of the time needed for learning.

3.3 Arithmetic Circuit Design

We compare the GA and RTS for arithmetic circuit design on the XC6216. We present the problem of designing a digital circuit in terms of a set of examples. Specifically we look at 2-bit adder and multiplier circuits, which have been considered of some interest for evolvable hardware approaches [Vassilev *et al.*, 1999][Miller *et al.*, 1998]. While we are concerned in the long term with designing circuits for pattern recognition, we use these arithmetic circuits to illustrate the effectiveness of automated design on a short time scale.

3.3.1 2-bit Multiplier

Using the 2-bit multiplier as a target, we show that we can learn circuits that perform nearly perfectly and use a similar amount of hardware resources to hand-tuned designs.

We use RTS for fitness optimization and allow 2.5×10^6 updates. No circuit performs perfectly, but the best performer made only 4 bit errors on 64 examples (1.56% error). The same performance was observed on a test set, indicating that the low error rate was not dependent on the initial circuit state, derived memory circuits (e.g. latches) or instabilities (e.g. oscillators). In fact, the output truth table given in table 3.3.1 indicates that instabilities are present in the circuit, but that their effects are minor.

Figure 3.3.1 shows the resulting layout. Only 11 function units are used in a 4×4 cell area. For comparison, the Xilinx multiplier macro uses 15 function units and wires in 16 cells (in a 4×5 rectangular footprint). Although the design constraints are different (the macro is intended to scale to any size multiplier), the learned solution demonstrates that adaptively designed circuits can be competitive in terms of hardware usage.

It should be noted that, while we would typically expect arithmetic circuits to be stable and error free, a good solution to a pattern recognition problem may allow for probabilistic outputs and errors much greater than 1.5% [Simard *et al.*, 1993]. Thus, while we do not expect this technique to redesign the multiplier, we have shown that it performs quite well on learning a circuit from a set of examples.

3.3.2 Effects of Dimensionality

In section 3.3.1 we saw that adaptive techniques can learn circuits from examples without a significant waste of chip resources. If we do not have a benchmark design available, however (as is the case in many pattern recognition problems), we are faced

	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0X10	0011
10	0000	0010	0100	0010
11	0000	0011	0110	1001

Table 3.1: Truth table for 2-bit multiplier solution

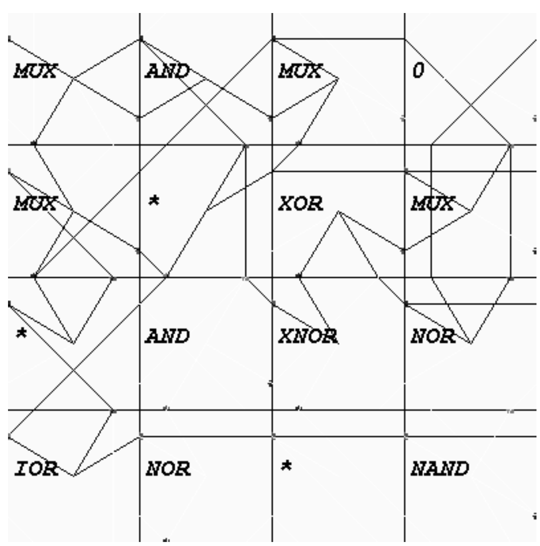


Figure 3.5: Learned circuit layout for 2-bit multiplier

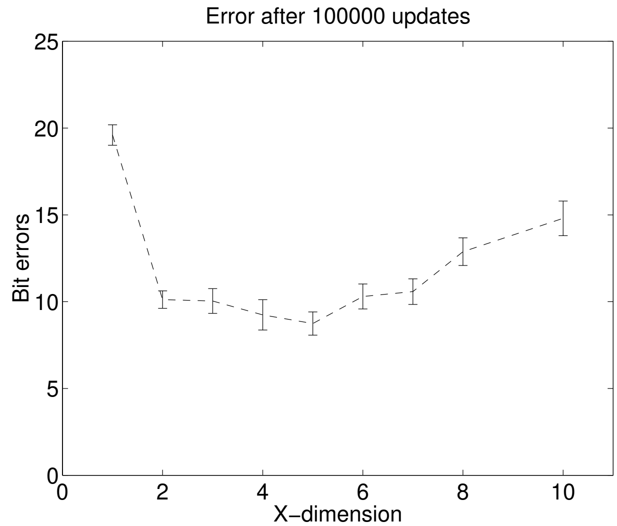


Figure 3.6: Mean training error levels achieved after 10^5 RTS updates

with the problem of selecting which chip resources to use. Allocating more cells to a problem increases the number of correct solutions, but also increases the dimension of the parameter space, making it more difficult to search.

Figure 3.3.2 illustrates this tradeoff for the multiplier problem, using RTS for 10^5 updates. In each case, a rectangular region of the chip was used with 4 cells in the y-dimension. The number of cells in the x-dimensions ranged from 1 to 10. With only 4 cells available, there is insufficient hardware to implement a solution to fit the data. As we add cells, the error for this short training run decreases steadily until we reach a 5×4 grid. Up to this point, the benefit from addition of gates outweighs the difficulty associated from increased dimension. As we add more gates, however, the error increases as the effects of dimensionality take over.

3.3.3 2-Bit Adder

We use the 2-bit adder to compare the relative performance of the RTS and the GA. We compare the performance in terms of similar execution time, since the GA involves more overhead for population dynamics. The population size for the GA is 100, so each generation for the involves checking 100 different circuits. Including

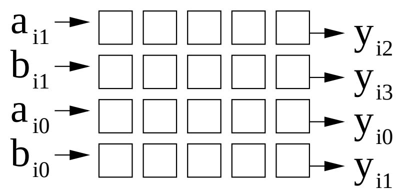


Figure 3.7: Chip resources used for 2-bit adder problem

overhead, the average execution time is approximately $111ms$. For RTS, we consider 22 possible changes, and each update takes approximately $18ms$. Thus, for a similar computation time, we run about 6 times as many updates for the RTS as generations of the GA. In terms of fitness evaluations, this results in a ratio of about 4:3 in favor of RTS.

We use a 5×4 grid of cells as the testbed for the problem, corresponding to a 360-dimensional configuration space. The inputs are presented as eastward inputs to the westernmost cells, and the outputs are the eastward outputs from the easternmost cells (see figure 3.3.3). The desired computation is presented as a set of examples, $\{(a_i, b_i), y_i\}_{i=1}^{64}$, where $a_i, b_i \in \{0,1\}^2$ and $y_i = a_i + b_i \in \{0,1\}^4$. Although the most significant bit of y_i will always be 0, we include it so that every output bit is completely specified by the input. As a type of symmetry hint, we allow permutations of the output bits. This slows fitness evaluation, but increases the space of correct designs, hopefully reducing search times.

We compare RTS for 3×10^5 updates and GA for 5×10^4 generations. This corresponds to approximately 5500 seconds of execution time, 6.6 million circuit evaluations for each RTS run and 5 million circuit evaluations for each run of the GA. Comparative results are given in table 3.3.3. Out of 25 test runs, the best GA performance on the presented example set was 41 bit errors of 256 possible (16%). While this is significantly better than a trivial all zero circuit, it is nowhere near performing the desired computation. In contrast, with the RTS, the mean error rate is only 4.28 bits (1.7%) on the training set, with average 16.6 average error bits (6.5%) on the test set.[2] In addition, the reactive tabu search gave one circuit that tested perfectly.

[2]Although the data sets used to train and test can be expected to contain the same data, the

Algorithm	Training		Test	
	Best	Mean	Best	Mean
RTS	0	4.28	0	16.6
GA	41	49.8	57	72.6
Zero	88	88	88	88
Random	128	128	128	128

Table 3.2: Resulting errors for the 2-bit adder problem

3.4 Conclusion

The results of section 3.3 demonstrate that we can design reasonably efficient and reliable circuits by hardware learning. While hardware evolution is known to be valuable, we have shown that, when we do not impose semantic constraints on the hardware learning model, use of a genetic algorithm may not be the best for fitness optimization.

Specifically, the reactive tabu search performs significantly better than the genetic algorithm for a 2-bit adder design for the same execution time. RTS also proved effective in designing a multiplier circuit that was $> 98\%$ correct and used fewer chip resources than the corresponding design provided by the manufacturer.

register states, evolved internal state, instabilities and particular data ordering (influencing, for example, state machine like operation) may cause the error on a particular training set to be artificially low. Resetting the configuration and testing on a 'new' data set should eliminate these artifacts.

Chapter 4 Conclusion

We have addressed two significant problems related to machine learning. First, in chapter 2 we addressed the problem of generalization and developed a model that allows analysis of a general learning problem. Second, in chapter 3 we addressed the problem of speed from the implementation standpoint. We showed that we can use reconfigurable hardware as a learning model with some success. While a similar approach has been around for some time in the context of evolvable hardware, we have shown that a the genetic algorithm is not necessarily the best.

Some significant questions remain related to both parts. In section 2.5 we introduced a method of applying the bin model analysis to a practical problem. It would be interesting to apply this to a real world data set. Also, some work is being done on investigating the generalization importance and correlations of individual data points, leading to an active learning strategy based on the bin model.

The learning hardware results presented here are for small, arithmetic problems. While these problems are easily analyzed, they are the least practical for this approach. It remains to use learning hardware to implement circuits that perform pattern recognition problems of larger size, those problems for which hand circuit design would be very difficult by conventional methods.

Appendix A Proofs of Results

Theorem 2.2.1 *The expected test error $\pi(\nu)$ is monotonically increasing in the empirical error ν.*

Proof of Theorem 2.2.1:

This is a special case of Theorem 2.3.1 with $\tilde{\pi}(g) = \pi(g) \forall g$.

■

Theorem 2.3.1 *For any $\tilde{\pi}(g)$, $p_{\mathcal{G}}(\cdot)$ and noise $\varepsilon(x)$, $\tilde{\pi}(\nu)$ is monotonically increasing in ν.*

Proof of Theorem 2.3.1:

Let $0 \leq k < N$. Then we compare $\tilde{\pi}(\frac{k}{N})$ and $\tilde{\pi}(\frac{k+1}{N})$.

$$\begin{aligned}
& \tilde{\pi}(\frac{k+1}{N}) - \tilde{\pi}(\frac{k}{N}) && \text{(A.1)} \\
= \; & \frac{\int_0^1 s\ P_{\nu|\tilde{\pi}}[\frac{k+1}{N}|s] p_{\tilde{\pi}}(s) ds}{\int_0^1 P_{\nu|\tilde{\pi}}[\frac{k+1}{N}|s] p_{\tilde{\pi}}(s) ds} - \frac{\int_0^1 t\ P_{\nu|\tilde{\pi}}[\frac{k}{N}|t] p_{\tilde{\pi}}(t) dt}{\int_0^1 P_{\nu|\tilde{\pi}}[\frac{k}{N}|t] p_{\tilde{\pi}}(t) dt} && \text{(A.2)} \\
= \; & \frac{\int s^{k+2}(1-s)^{N-k-1} p_{\tilde{\pi}}(s) ds}{\int s^{k+1}(1-s)^{N-k-1} p_{\tilde{\pi}}(s) ds} - \frac{\int t^{k+1}(1-t)^{N-k} p_{\tilde{\pi}}(t) dt}{\int t^{k}(1-t)^{N-k} p_{\tilde{\pi}}(t) dt} && \text{(A.3)} \\
\propto \; & \iint s^{k+2}(1-s)^{N-k-1} t^k (1-t)^{N-k} p_{\tilde{\pi}}(t) p_{\tilde{\pi}}(s) ds\ dt \\
& - \iint t^{k+1}(1-t)^{N-k} s^{k+1}(1-s)^{N-k-1} p_{\tilde{\pi}}(t) p_{\tilde{\pi}}(s) ds\ dt && \text{(A.4)} \\
= \; & \iint s(s-t)(1-t) s^k t^k (1-s)^{N-k-1}(1-t)^{N-k-1} p_{\tilde{\pi}}(t) p_{\tilde{\pi}}(s) ds\ dt && \text{(A.5)} \\
= \; & \iint t(t-s)(1-s) t^k s^k (1-t)^{N-k-1}(1-s)^{N-k-1} p_{\tilde{\pi}}(s) p_{\tilde{\pi}}(t) dt\ ds && \text{(A.6)} \\
= \; & \frac{1}{2} \iint (s(s-t)(1-t) + t(t-s)(1-s)) \\
& \cdot t^k s^k (1-t)^{N-k-1}(1-s)^{N-k-1} p_{\tilde{\pi}}(s) p_{\tilde{\pi}}(t) dt\ ds && \text{(A.7)} \\
= \; & \frac{1}{2} \iint (s-t)^2 t^k s^k (1-t)^{N-k-1}(1-s)^{N-k-1} p_{\tilde{\pi}}(s) p_{\tilde{\pi}}(t) dt\ ds && \text{(A.8)} \\
\geq \; & 0 && \text{(A.9)}
\end{aligned}$$

The constant of proportionality in (A.4) is positive, (A.6) is (A.5) rewritten with a change of variables, and (A.7) is obtained by summing (A.6) and (A.5). Thus $\tilde{\pi}(0) \leq \tilde{\pi}(\frac{1}{N}) \leq \tilde{\pi}(\frac{2}{N}) \leq \ldots \leq \tilde{\pi}(1)$ completing the proof.

■

Theorem 2.3.2 *If $\varepsilon(x)$ is regular, then the following are equivalent:*

- *$\tilde{\pi}(\pi)$ is monotonically increasing (decreasing) in π*
- *for any $p_\pi(\cdot)$, $\pi(\nu)$ is monotonically increasing (decreasing) in ν*

Proof of Theorem 2.3.2:

The proof is similar to that of Theorem 2.3.1. Let $0 \leq k < N$. We look at $\pi(\frac{k+1}{N}) - \pi(\frac{k}{N})$.

$$\begin{aligned}
& \pi(\frac{k+1}{N}) - \pi(\frac{k}{N}) \\
= \;& \frac{\int_0^1 s\ P_{\nu|\pi}[\frac{k+1}{N}|s]p_\pi(s)ds}{\int_0^1 P_{\nu|\pi}[\frac{k+1}{N}|s]p_\pi(s)ds} - \frac{\int_0^1 t\ P_{\nu|\pi}[\frac{k}{N}|t]p_\pi(t)dt}{\int_0^1 P_{\nu|\pi}[\frac{k}{N}|t]p_\pi(t)dt} && \text{(A.10)} \\
= \;& \frac{\int s\ \tilde{\pi}(s)^{k+1}(1-\tilde{\pi}(s))^{N-k-1}p_\pi(s)ds}{\int \tilde{\pi}(s)^{k+1}(1-\tilde{\pi}(s))^{N-k-1}p_\pi(s)ds} - \frac{\int t\ \tilde{\pi}(t)^{k}(1-\tilde{\pi}(t))^{N-k}p_\pi(t)dt}{\int \tilde{\pi}(t)^{k}(1-\tilde{\pi}(t))^{N-k}p_\pi(t)dt} && \text{(A.11)} \\
\propto \;& \iint s\ \tilde{\pi}(s)^{k+1}(1-\tilde{\pi}(s))^{N-k-1}\tilde{\pi}(t)^k(1-\tilde{\pi}(t))^{N-k}p_\pi(t)p_\pi(s)ds\ dt \\
& - \iint t\ \tilde{\pi}(t)^k(1-\tilde{\pi}(t))^{N-k}\tilde{\pi}(s)^{k+1}(1-\tilde{\pi}(s))^{N-k-1}p_\pi(t)p_\pi(s)ds\ dt && \text{(A.12)} \\
= \;& \iint (s-t)\tilde{\pi}(s)(1-\tilde{\pi}(t))\tilde{\pi}(s)^k\tilde{\pi}(t)^k \\
& \cdot (1-\tilde{\pi}(s))^{N-k-1}(1-\tilde{\pi}(t))^{N-k-1}p_\pi(t)p_\pi(s)ds\ dt && \text{(A.13)} \\
= \;& \iint (t-s)\tilde{\pi}(t)(1-\tilde{\pi}(s))\tilde{\pi}(s)^k\tilde{\pi}(t)^k \\
& \cdot (1-\tilde{\pi}(s))^{N-k-1}(1-\tilde{\pi}(t))^{N-k-1}p_\pi(t)p_\pi(s)ds\ dt && \text{(A.14)} \\
= \;& \frac{1}{2}\iint (s-t)(\tilde{\pi}(s)-\tilde{\pi}(t))t^k s^k \\
& \cdot (1-t)^{N-k-1}(1-s)^{N-k-1}p_\pi(s)p_\pi(t)dt\ ds && \text{(A.15)}
\end{aligned}$$

The constant of proportionality in (A.12) is positive, and the steps are essentially the same as those in the proof of Theorem 2.3.1.

If $\tilde{\pi}(\pi)$ is monotonically increasing in π, then $(s-t)(\tilde{\pi}(s)-\tilde{\pi}(t)) \geq 0 \forall s,t$, the integrand in (A.15) is always nonnegative, and hence $\pi(\nu)$ is monotonically increasing in ν. Likewise, if $\tilde{\pi}(\pi)$ is monotonically decreasing in π, then the integrand in (A.15) is always nonpositive and $\pi(\nu)$ is monotonically decreasing in ν.

Now suppose $\tilde{\pi}(\pi)$ is not monotonically increasing in π. Then $\exists \pi_1, \pi_2$ such that $0 < \pi_1 < \pi_2 < 1$ and $\tilde{\pi}(\pi_1) > \tilde{\pi}(\pi_2)$. Then taking $p_\pi(\pi) = \frac{1}{2}(\delta(\pi - \pi_1) + \delta(\pi - \pi_2))$, (A.15) becomes

$$\begin{aligned}
&\pi(\frac{k+1}{N}) - \pi(\frac{k}{N}) \\
&\propto \ \frac{1}{2}\Big((\pi_1 - \pi_2)(\tilde{\pi}(\pi_1) - \tilde{\pi}(\pi_2))\pi_1^k\pi_2^k(1-\pi_1)^{N-k-1}(1-\pi_2)^{N-k-1} \\
&\qquad +(\pi_2 - \pi_1)(\tilde{\pi}(\pi_2) - \tilde{\pi}(\pi_1))\pi_1^k\pi_2^k(1-\pi_1)^{N-k-1}(1-\pi_2)^{N-k-1}\Big) \qquad \text{(A.16)} \\
&< \ 0 \qquad \text{(A.17)}
\end{aligned}$$

since $\pi_1 - \pi_2 < 0$, $\tilde{\pi}(\pi_1) - \tilde{\pi}(\pi_2) > 0$ and $\pi_1^k\pi_2^k(1-\pi_1)^{N-k-1}(1-\pi_2)^{N-k-1} > 0$.

Hence, if $\tilde{\pi}(\pi)$ is not monotonically increasing in π, then there is a p_π for which $\pi(\nu)$ fails to be monotonically increasing in ν. Likewise, if $\tilde{\pi}(\pi)$ is not monotonically decreasing in π, then in the same way we can find a p_π for which $\pi(\nu)$ fails to be monotonically decreasing in ν. Thus we have established equivalence proving the theorem.

■

Theorem 2.4.1 *If $p_{\nu|\pi}$ is non-zero, continuous and differentiable on $(\nu, \pi) \in \mathcal{E}^2$ then $\pi(\nu)$ is monotonically increasing in ν for any p_π iff $(\frac{\partial}{\partial\nu}p_{\nu|\pi})/p_{\nu|\pi}$ is monotonically increasing in π.*

Proof of Theorem 2.4.1:

$$\begin{aligned}
\pi(\nu) &= \frac{\int s p_{\nu|\pi}(\nu|s)p_\pi(s)ds}{\int p_{\nu|\pi}(\nu|s)p_\pi(s)ds} \qquad \text{(A.18)} \\
\frac{d\pi(\nu)}{d\nu} &= \frac{1}{\left(\int p_{\nu|\pi}(\nu|s)p_\pi(s)ds\right)^2}\Bigg(\int s\frac{d}{d\nu}p_{\nu|\pi}(\nu|s)p_\pi(s)ds \int p_{\nu|\pi}(\nu|t)p_\pi(t)dt \\
&\quad - \int s p_{\nu|\pi}(\nu|s)p_\pi(s)ds \int \frac{d}{d\nu}p_{\nu|\pi}(\nu|t)p_\pi(t)ds\Bigg) \qquad \text{(A.19)} \\
&\propto \iint s\frac{d}{d\nu}p_{\nu|\pi}(\nu|s)p_{\nu|\pi}(\nu|t)p_\pi(s)p_\pi(t)ds\ dt
\end{aligned}$$

$$
\begin{aligned}
& - \iint s p_{\nu|\pi}(\nu|s) \frac{d}{d\nu} p_{\nu|\pi}(\nu|t) p_\pi(s) p_\pi(t) ds\ dt && \text{(A.20)} \\
= \quad & \iint s \left(\frac{d}{d\nu} p_{\nu|\pi}(\nu|s) p_{\nu|\pi}(\nu|t) \right. \\
& \left. - \ p_{\nu|\pi}(\nu|s) \frac{d}{d\nu} p_{\nu|\pi}(\nu|t) \right) p_\pi(s) p_\pi(t) ds\ dt && \text{(A.21)} \\
= \quad & \iint t \left(\frac{d}{d\nu} p_{\nu|\pi}(\nu|t) p_{\nu|\pi}(\nu|s) \right. \\
& \left. - \ p_{\nu|\pi}(\nu|t) \frac{d}{d\nu} p_{\nu|\pi}(\nu|s) \right) p_\pi(s) p_\pi(t) ds\ dt && \text{(A.22)} \\
= \quad & \frac{1}{2} \iint (s-t) \left(\frac{d}{d\nu} p_{\nu|\pi}(\nu|s) p_{\nu|\pi}(\nu|t) \right. \\
& \left. - \ p_{\nu|\pi}(\nu|s) \frac{d}{d\nu} p_{\nu|\pi}(\nu|t) \right) p_\pi(s) p_\pi(t) ds\ dt && \text{(A.23)}
\end{aligned}
$$

The constant of proportionality in (A.20) is always positive and (A.22) is (A.21) rewritten with a change of variables. Combining (A.21) and (A.22) gives (A.23).

First we show sufficiency. For any $s > t$, the integrand in (A.23) is nonnegative if

$$
\begin{aligned}
\frac{d}{d\nu} p_{\nu|\pi}(\nu|s) p_{\nu|\pi}(\nu|t) \quad &\geq \quad p_{\nu|\pi}(\nu|s) \frac{d}{d\nu} p_{\nu|\pi}(\nu|t) && \text{(A.24)} \\
\frac{\frac{d}{d\nu} p_{\nu|\pi}(\nu|s)}{p_{\nu|\pi}(\nu|s)} \quad &\geq \quad \frac{\frac{d}{d\nu} p_{\nu|\pi}(\nu|t)}{p_{\nu|\pi}(\nu|t)} && \text{(A.25)}
\end{aligned}
$$

Since $p_{\nu|\pi}$ is assumed to be continuous, this is equivalent to increasing monotonicity of $(\frac{d}{d\nu} p_{\nu|\pi})/p_{\nu|\pi}$ in π.

To show necessity, suppose that $(\frac{d}{d\nu} p_{\nu|\pi})/p_{\nu|\pi}$ is not monotonically increasing in π. Then $\exists \pi_1, \pi_2, \nu_0$ such that

$$
\begin{aligned}
\pi_1 \quad &> \quad \pi_2 && \text{(A.26)} \\
\frac{\frac{d}{d\nu} p_{\nu|\pi}(\nu_0|\pi_1)}{p_{\nu|\pi}(\nu_0|\pi_1)} \quad &< \quad \frac{\frac{d}{d\nu} p_{\nu|\pi}(\nu_0|\pi_2)}{p_{\nu|\pi}(\nu_0|\pi_2)} && \text{(A.27)}
\end{aligned}
$$

Then letting $p_\pi(\pi) = \frac{1}{2}(\delta(\pi - \pi_1) + \delta(\pi - \pi_2))$, we get

$$\begin{aligned}
&\frac{d\pi(\nu)}{d\nu} \\
&\propto \frac{1}{2}\iint (s-t)\left(\frac{d}{d\nu}p_{\nu|\pi}(\nu|s)p_{\nu|\pi}(\nu|t)\right. \\
&\qquad \left. - p_{\nu|\pi}(\nu|s)\frac{d}{d\nu}p_{\nu|\pi}(\nu|t)\right) p_\pi(s)p_\pi(t)ds\ dt && \text{(A.28)} \\
&= \frac{1}{4}(\pi_1-\pi_2)(\frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi_1)p_{\nu|\pi}(\nu|\pi_2) - \frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi_2)p_{\nu|\pi}(\nu|\pi_1)) && \text{(A.29)} \\
& && \text{(A.30)} \\
&= \frac{1}{4}(\pi_1-\pi_2)\left(\frac{\frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi_1)}{p_{\nu|\pi}(\nu|\pi_1)} - \frac{\frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi_2)}{p_{\nu|\pi}(\nu|\pi_2)}\right)p_{\nu|\pi}(\nu|\pi_1)p_{\nu|\pi}(\nu|\pi_2) && \text{(A.31)} \\
&< 0 && \text{(A.32)}
\end{aligned}$$

Thus $\pi(\nu)$ fails to be monotonically increasing at ν_0 and the proof is complete.

■

Theorem 2.4.2 *If $\sigma(g) = \sigma(\pi(g))\ \ \forall g \in \mathcal{G}$, then in the large data limit $N \to \infty$, $\pi(\nu)$ is monotonically increasing in ν for all p_π iff $\frac{e_{max}-\pi_1}{e_{max}-\pi_2} \leq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \leq \frac{\pi_1-e_{min}}{\pi_2-e_{min}}$ whenever $\pi_1 > \pi_2$.*

Proof of Theorem 2.4.2:

In the large data limit

$$p_{\nu|\pi}(\nu|\pi) \propto \exp(-\frac{N(\nu-\pi)^2}{2\sigma(\pi)^2}) \tag{A.33}$$

$$\frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi) = \frac{N(\pi-\nu)}{\sigma(\pi)^2}p_{\nu|\pi}(\nu|\pi) \tag{A.34}$$

Theorem 2.4.1 tells us that increasing monotonicity of $\pi(\nu)$ in ν for any p_π is equivalent to increasing monotonicity of $(\frac{d}{d\nu}p_{\nu|\pi})/p_{\nu|\pi}$ in π.

$$\frac{\frac{d}{d\nu}p_{\nu|\pi}(\nu|\pi)}{p_{\nu|\pi}(\nu|\pi)} = \frac{N(\pi-\nu)}{\sigma(\pi)^2} \tag{A.35}$$

Thus we want, $\forall \nu \in \mathcal{E}$, if $\pi_1 > \pi_2$, then

$$\frac{N(\pi_1 - \nu)}{\sigma(\pi_1)^2} \geq \frac{N(\pi_2 - \nu)}{\sigma(\pi_2)^2} \tag{A.36}$$

We consider three possible cases. If $e_{min} \leq \nu < \pi_2$, then we have monotonicity iff

$$\frac{N(\pi_1 - \nu)}{\sigma(\pi_1)^2} \geq \frac{N(\pi_2 - \nu)}{\sigma(\pi_2)^2} \tag{A.37}$$

$$\frac{\pi_1 - \nu}{\pi_2 - \nu} \geq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \tag{A.38}$$

The worst case is $\nu = e_{min}$, so it is sufficient that

$$\frac{\pi_1 - e_{min}}{\pi_2 - emin} \geq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \tag{A.39}$$

If $\pi_2 \leq \nu < \pi_1$, then

$$\frac{N(\pi_1 - \nu)}{\sigma(\pi_1)^2} > 0 \geq \frac{N(\pi_2 - \nu)}{\sigma(\pi_2)^2} \tag{A.40}$$

and we always have monotonicity.

If $\pi_1 \leq \nu \leq e_{max}$, then we have monotonicity iff

$$\frac{N(\pi_1 - \nu)}{\sigma(\pi_1)^2} \geq \frac{N(\pi_2 - \nu)}{\sigma(\pi_2)^2} \tag{A.41}$$

$$\frac{\nu - \pi_1}{\nu - \pi_2} \leq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \tag{A.42}$$

The worst case is $\nu = e_{max}$, so it is sufficient that

$$\frac{e_{max} - \pi_1}{e_{max} - \pi_2} \leq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \tag{A.43}$$

(A.39) and (A.43) tell us that $(\frac{d}{d\nu} p_{\nu|\pi})/p_{\nu|\pi}$ is monotonically increasing in π for all ν iff $\frac{e_{max}-\pi_1}{e_{max}-\pi_2} \leq \frac{\sigma(\pi_1)^2}{\sigma(\pi_2)^2} \leq \frac{\pi_1 - e_{min}}{\pi_2 - e_{min}}$ whenever $\pi_1 > \pi_2$, proving the theorem.

$\blacksquare$

Proposition A.0.1 *For a regression problem, assume a squared error measure and assume additive zero-mean Gaussian noise.*

$$
\begin{aligned}
e(g,x) &= (g(x)-f(x))^2 && \text{(A.44)}\\
\tilde{f}(x) &= f(x)+\eta && \text{(A.45)}\\
\eta &\sim \mathcal{N}(0,\sigma_\eta^2) && \text{(A.46)}\\
\tilde{e}(g,x) &= (g(x)-\tilde{f}(x))^2 && \text{(A.47)}
\end{aligned}
$$

If the noiseless errors e are distributed like $p_{e|g}$ and the noisy errors $\tilde{e}$ are distributed like $p_{\tilde{e}}$, then $p_{e|g}$ and $p_{\tilde{e}}$ are related by

$$
\widehat{p_{\tilde{e}|g}}(\omega) = \frac{\widehat{p_{e|g}}\left(\frac{\omega}{1+2i\sigma_\eta^2\omega}\right)}{\sqrt{1+2i\sigma_\eta^2\omega}} \qquad \text{(A.48)}
$$

Proof of Proposition A.0.1:

Let $s(x)=g(x)-f(x)$.

$$
\begin{aligned}
e(g,x) &= s(x)^2 && \text{(A.49)}\\
\tilde{e}(g,x) &= (s(x)-\eta)^2 && \text{(A.50)}\\
\widehat{p_{\tilde{e}|g}}(\omega) &= E_{\tilde{e}}[\exp(-i\omega\tilde{e})|g] && \text{(A.51)}\\
&= E_{x,\eta}[\exp(-i\omega\tilde{e}(x,g)] && \text{(A.52)}\\
&= \iint \exp\left(-i\omega\left(s(x)-\eta\right)^2\right) p_{e|g}(s(x)^2)p_\eta(\eta)d\eta\ dx && \text{(A.53)}\\
&= \frac{1}{\sqrt{2\pi}\sigma_\eta}\int \exp\left(-i\omega s(x)^2\right) p_{e|g}(s(x)^2)\cdot \\
&\quad \int \exp\left(-i\omega\left(\eta^2-2s(x)\eta\right)\right)\exp\left(-\frac{\eta^2}{2\sigma_\eta^2}\right) d\eta\ dx && \text{(A.54)}\\
&= \frac{1}{\sqrt{2\pi}\sigma_\eta}\int \exp\left(-i\omega s(x)^2\right) p_{e|g}(s(x)^2)\cdot \\
&\quad \int \exp\left(-\left(i\omega+\frac{1}{2\sigma_\eta^2}\right)\left(\eta^2-\frac{4i\omega\sigma_\eta^2 s(x)\eta}{2i\omega\sigma_\eta^2+1}\right)\right) d\eta\ dx && \text{(A.55)}\\
&= \frac{1}{\sqrt{2\pi}\sigma_\eta}\int \exp\left(-i\omega s(x)^2\right) p_{e|g}(s(x)^2)\exp\left(-\frac{2\omega^2\sigma_\eta^2 s(x)^2}{2i\omega\sigma_\eta^2+1}\right)\cdot
\end{aligned}
$$

$$\int \exp\left(-\left(i\omega+\frac{1}{2\sigma_\eta^2}\right)\left(\eta-\frac{2i\omega\sigma_\eta^2 s(x)}{2i\omega\sigma_\eta^2+1}\right)^2\right) d\eta\ dx \tag{A.56}$$

$$\begin{aligned}
&= \frac{1}{\sqrt{2\pi}\sigma_\eta}\int \exp\left(-i\omega s(x)^2\right) p_{e|g}(s(x)^2)\exp\left(-\frac{2\omega^2\sigma_\eta^2 s(x)^2}{2i\omega\sigma_\eta^2+1}\right) \\
&\quad \cdot \int \exp\left(-\left(i\omega+\frac{1}{2\sigma_\eta^2}\right)\left(\eta-\frac{2i\omega\sigma_\eta^2 s(x)}{2i\omega\sigma_\eta^2+1}\right)^2\right) d\eta\ dx
\end{aligned} \tag{A.57}$$

$$\begin{aligned}
&= \frac{1}{\sqrt{2\pi}\sigma_\eta}\int \exp\left(-is(x)^2\left(\omega-\frac{2i\omega^2\sigma_\eta^2}{2i\omega\sigma_\eta^2+1}\right)\right) \\
&\quad \cdot p_{e|g}(s(x)^2)\sqrt{\frac{2\pi\sigma_\eta^2}{2i\omega\sigma_\eta^2+1}}dx
\end{aligned} \tag{A.58}$$

$$= \frac{1}{2i\omega\sigma_\eta^2+1}\int \exp\left(-i\left(\frac{\omega}{2i\omega\sigma_\eta^2+1}\right)s(x)^2\right)p_{e|g}(s(x)^2)dx \tag{A.59}$$

$$= \frac{E_x[\exp\left(-i\left(\frac{\omega}{1+2i\sigma_\eta^2\omega}\right)e(g|x)\right)]}{\sqrt{1+2i\sigma_\eta^2\omega}} \tag{A.60}$$

$$= \frac{\widehat{p_{e|g}}\left(\frac{\omega}{1+2i\sigma_\eta^2\omega}\right)}{\sqrt{1+2i\sigma_\eta^2\omega}} \tag{A.61}$$

■

Bibliography

[Abu-Mostafa and Song, 1996] Y. Abu-Mostafa and X. Song. Bin model for neural networks. In *Proceedings of ICONIP'96*, pages 169–173, Hong Kong, 1996.

[Abu-Mostafa, 1989] Y. Abu-Mostafa. The vapnik-chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1:312–317, 1989.

[Akaike, 1970] H. Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22:203, 1970.

[Battiti and Tecchiolli, 1994] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal*, 6(2):126–140, 1994.

[Battiti and Tecchiolli, 1995] R. Battiti and G. Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, 1995.

[Bennett *et al.*, 1999] F.H. Bennett, J.R. Koza, M.A. Keane, J. Yu, W. Mydlowec, and O. Stiffelman. Evolution by means of genetic programming of analog circuits that perform digital functions. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1477–1483, San Francisco, 1999. Morgan Kaufmann.

[Bishop, 1995a] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, 1995.

[Bishop, 1995b] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[Cataltepe *et al.*, 1999] Z. Cataltepe, Y. S. Abu-Mostafa, and M. Magdon-Ismail. No free lunch for early stopping. *Neural Computation*, 11:995–1009, 1999.

[Cover and Thomas, 1991] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Series in Telecomunications. Wiley, 1991.

[Duda and Hart, 1973] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. J. Wiley & Sons, 1973.

[Goldberg, 1989] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, Ma., 1989.

[Haykin, 1994] S. Haykin. *Neural Networks*. Macmillan College Publishing Company, New York, 1994.

[Higuchi *et al.*, 1999] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.

[Keymeulen *et al.*, 1998] D. Keymeulen, K. Konaka, M. Iwata, Y. Kuniyoshi, and T. Higuchi. Robot learning using gate-level evolvable hardware. In *Proceedings of the Sixth European Workshop on Learning Robots*, New York, 1998. Springer-Verlag.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Koza *et al.*, 1998] J.R. Koza, F.H. Bennett, J.L. Hutchings, S.L. Bade, M.A. Keane, and D. Andre. Evolving computer programs using rapidly reconfigurable field-programmable gate arrays and genetic programming. In *Proceedings of the ACM Sixth International Symposium on Field Programmable Gate Arrays*, pages 209–219, New York, 1998. ACM Press.

[Magdon-Ismail *et al.*, 1998] M. Magdon-Ismail, A. Nicholson, and Y. S. Abu-Mostafa. Financial markets: very noisy information processing. *Proceedings of the IEEE*, 86(11):2184–2195, 1998.

[Mead, 1989] C. Mead. *Analog VLSI and Neural Systems.* Addison-Wesley, Reading, MA, 1989.

[Miller *et al.*, 1998] J.F. Miller, P. Thompson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*, New York, 1998. John Wiley & Sons, Inc.

[Miller, 1999] J. Miller. On the filtering properties of evolved gate arrays. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 2–11, Los Alamitos, CA, 1999. IEEE Computer Society.

[Moody, 1991] J. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 4, pages 847–854, 1991.

[Perkowski *et al.*, 1999] M. Perkowski, A. Chebotarev, and A. Mishchenko. Evolvable hardware or learning hardware? induction of state machines from temporal logic constraints. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 129–138, Los Alamitos, CA, 1999. IEEE Computer Society.

[Ramacher and Ruckert, 1991] U. Ramacher and U. Ruckert, editors. *VLSI Design of Neural Networks.* Kluwer Academic Publishers, Boston, 1991.

[Rumelhart *et al.*, 1986] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[Schewel, 1997] J. Schewel. *A Hardware/Software Co-Design System using Configurable Computing Technology.* Virtual Computer Corp., 1997.

[Schwartz *et al.*, 1990] D.B. Schwartz, V.K. Samalam, S.A. Solla, and J.S. Denker. Exhaustive learning. *Neural Computation*, 2(2):374–385, 1990.

[Simard *et al.*, 1993] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, volume 4, pages 50–58. Morgan Kaufmann, 1993.

[Solla, 1992] S.A. Solla. Supervised learning: A theoretical framework. In M. Casdagli and S. Eubank, editors, *Nonlinear Modelling and Forecasting*, pages 25–38, 1992.

[Stone, 1974] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society Series B*, 36:111–147, 1974.

[Thompson and Layzell, 1999] A. Thompson and P. Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, 1999.

[Thompson, 1998] A. Thompson. *Hardware evolution: automatic design of electronic circuits in reconfigurable hardware by Artificial Evolution.* Springer-Verlag, New York, 1998.

[Vapnik and Chervonenkis, 1971] V. N. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[Vapnik, 1998] V.N. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, Inc., New York, 1998.

[Vassilev *et al.*, 1999] V.K. Vassilev, J.F. Miller, and T.C. Fogarty. On the nature of two-bit multiplier landscapes. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 36–45, Los Alamitos, CA, 1999. IEEE Computer Society.

[Wolpert, 1996] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[Xilinx, Inc., 1997] Xilinx, Inc. *XC6200 Field Programmable Gate Arrays Data Sheet.* 1997.