INCREMENTAL TRAINING AND GROWTH OF ARTIFICIAL NEURAL NETWORKS

BY

YANG LI

B.A., University of Illinois at Urbana-Champaign, 2007

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Adviser:

Associate Professor Mark Hasegawa-Johnson

ABSTRACT

Training of automatic pattern recognition or function regression systems has been investigated for decades, and it is fairly well understood that the usage of limited amounts of empirical data in the training of such systems necessarily leads to generalization difficulties. The focus of this work is to investigate the generalization issues associated with a particular class of estimators - artificial neural networks - and formulate a novel method to improve the trade-off between performance and generalizability when it comes to training with a limited amount of empirical data. The improvement comes from an effective utilization of prior knowledge: if a network can be trained on a large training corpus sharing certain characteristics with the data from the task at hand, then the network can be "grown" to be adapted to solve the current task. The network carries the structure obtained from its training on the large dataset over to the smaller dataset; if there are similarities in the structure, this preservation of structure across applications expedites training and ensures lower variability. The thesis presents a theoretical argument on the relationship between bounds of total expected error based on the number of trainable parameters in a network, develops a method to perform said growths, and conducts two series of experiments to demonstrate the superior generalizability of the grown networks.

To Life, the greatest mystery of all.

ACKNOWLEDGMENTS

First and foremost, this work could not have come to be were it not for the guidance from my adviser, Prof. Mark Hasegawa-Johnson. Many a time was I lost in the labyrinth of equations and inequalities until he showed me the meaning of it all. I would also like to express my deepest gratitude to my parents, who made and trained me from scratch, so that I could have come this far. Lastly, Lethe my love, thank you for being *ma raison d'etrè* during the darkest times. It was enough knowing you were somewhere, striving for that same future. Thank you.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

For a wide range of automatic recognition and classification, control systems, and machine learning applications, where general adaptive pattern recognition or function regression is required, artificial neural networks (ANNs) have become quite popular, due to their versatility in their universal approximating capabilities [1]. However, along with this increased power comes increased system complexity and adaptation difficulty. For example, constructing a three-layer time-delayed ANN (TDNN) with 1024 inputs would require approximately $1025(N + 1)$ parameters, assuming there are N nodes in the hidden layer, whereas constructing a least mean squared (LMS) adaptive filter only requires 1024 parameters. The ability of the TDNN to approximate a wider range of functions comes at a cost of more complex structures and more parameters. Not only does this increased parametric dimensionality increases training and testing time, but it also adversely affects the convergence rate of the system, given finite-sized training datasets [2]. On the other hand, it is obvious that the more nodes (thus, more parameters) an ANN contains, the greater its approximating power. Because of this, the selection of network structures has always been an art, requiring from the designer a priori knowledge of both the problem domain and the amount of data available for the training of the network. There have been several works on the bounds of convergence rates of ANNs, analyzing the relationship between the size of the training dataset and the estimation error of the ANN (see [3], [4], [5], and [6]). While these works often only give order-of-magnitude bounds on specific classes of ANNs, it can be seen that the error bound often shares an approximately log-inverse relationship with the number of parameters in the ANN. Therefore, it becomes clear that a way of reducing total error, regardless of application, is to find a way to reduce the number of parameters needed in an ANN, while maintaining the approximating accuracy of the ANN for the class of functions pertaining to the application (this

can be empirically measured through the training error). In the probably-approximately-correct (PAC) framework, this trade-off is represented by the bias-variance trade-off of the classifier: while a classifier with more complex structure and a larger number of parameters can approximate more closely the discriminating function, when trained with a finite number of samples, the convergence of the classifier with respect to the amount of training data will be slowed; i.e., given the same amount of training data, the classifier with more parameters to adapt will have a higher error variance than a classifier with fewer adapting parameters.

This limitation has led researchers to investigate the nature of this trade-off. A number of important results have been formulated both in the derivation of convergence rate and error variance bounds, and in the development of practical implementations of ANN systems to automatically and algorithmically determine a good balance in the bias/variance trade-off (see [2], [5], and [7]). This thesis proposes a new method of ANN training, aiming to address this issue from a pragmatic point of view: through the proposed method, an ANN can be "grown" from an existing ANN, with the new ANN mathematically related to the original in a straightforward and meaningful way. The advantages of this method are two-fold: one can train ANN one node at a time, reducing training memory usage; and more significantly, this growth scheme allows previously trained ANNs to be expanded, resulting in a systematic method to effectively utilize a priori knowledge.

The remaining chapters of the thesis are organized as follows: Chapter 2 offers a thorough survey of the background of the problem, including the theoretical roots of the bias-variance problem, convergence bounds, and various methods proposed to achieve desirable trade-offs. Chapter 3 details the method for "growing" the ANNs, while explicating the effects of different kinds of growths on the actual functions implemented by the grown ANN. It also gives theoretical justification for the relationship between estimation error bound and parametric dimensionality when applied specifically to this class of ANNs. Chapter 4 describes a series of simulated experiments performed on grown ANNs to demonstrate its convergence properties in comparison with normal ANNs. Lastly, Chapter 5 summarizes the results and presents the final conclusions.

2 BACKGROUND AND MOTIVATION

2.1 Estimation of Dependencies

In Vapnik's seminal work *Estimation of Dependencies Based on Empirical Data* [2], he posed the problem of expected risk minimization based on empirical data. In both pattern recognition and regression tasks, the goal of the estimation system is to minimize the overall expected risk, related to the error through some loss function. For the problem of pattern recognition, this is a function of the difference between the estimated and the actual class-assignment given to each observation. For the problem of regression, it is a function of the difference between the estimated and the actual mean of the random variable. Regardless of context, the minimization of total risk is always based on the minimization of empirical risk, that is, the loss function evaluated on the classification/regression error given by a collection of empirical samples. This begs the question: Does the estimate obtained through the minimization of the empirical risk come close to the function which minimizes the true total risk?

The answer is positive under the condition of uniform convergence of frequency of events to their probabilities (pattern recognition context), or uniform convergence of means to mathematical expectations (regression context).

2.1.1 The problem of pattern recognition

Mathematically, the problem of pattern recognition is to identify, within a predefined class of indicator functions, the best indicator function which, given an observation, would accurately find its true classification, among several classes. This class of functions can be denoted by $F(x, \alpha)$, where x is the observation, and α represents the adjustable parameters for the class of functions F. If the true classification for an observation x_i is y_i, then the

minimization of empirical risk can be expressed as

$$\hat{\alpha}_{emp} = \arg\min_{\alpha}(\sum_{i=1}^{N}(y_i - F(x_i, \alpha))^2) \tag{2.1}$$

Compare this with the minimization of the total expected risk:

$$\hat{\alpha} = \arg\min_{\alpha}(\int_{-\infty}^{\infty}(y - F(x, \alpha))^2 P(x, y)dxdy) \tag{2.2}$$

where $P(x, y)$ is the joint probability density function (pdf) of x and y.

2.1.2 The problem of regression

The problem of regression is defined to be the estimation of the mean of the output y, conditioned upon input x. This can be written as

$$F(x) = \int yP(y|x)dx \tag{2.3}$$

where $P(y|x)$ is the conditional pdf of y given x. Similar to the problem of pattern recognition, if $F(x, \alpha)$ is the class of functions used to estimate the stochastic denpendencies between x and y, then the minimization of empirical and total risks can be written in the exact same forms as Eqs. (2.1) and (2.2). The only difference is that $F(x, \alpha)$ is no longer constrained to be indicator functions.

2.1.3 Uniform convergence

The law of large numbers indicates that given an infinite amount of samples, the frequencies of events converge to their probabilities; similarly, the mean of the samples converges to the mathematical expectation of the sampled random variable. This provides theoretical justification of using empirical risk as a suitable substitute for total risk when one has infinitely large datasets. But how does the empirical risk behave when one only possesses a finite amount of data?

This question cannot be answered without further assumptions. For instance, if a pattern recognition system is constructed from l known samples such that for any testing input that matches one of the samples, it

gives the correct output given by the sample, but for any testing input that does not match any known sample, it always gives 0, then it will clearly have a large total risk, while always having zero empirical risk. It is true that with an infinite amount of data, all possible inputs can be matched, and the total risk will approach zero; but there is no conclusive way to determine the finite-sample behavior of this pattern recognition system, due to the nonuniform nature of the convergence of events to probabilities (in this case, the convergence of the classification by the indicator function to the true underlying classification of the input observation).

However, if uniform convergence of frequencies to probability, or sample-mean to expectation, can be assumed, then it becomes possible to establish a bound on the convergence rate of the empirical risk to the total risk. This becomes tractable when there is a finite number of N functions within the class of functions used to estimate the dependency, or when the class of functions can be covered by an ϵ-net of a finite class of $N(\epsilon)$ functions. With these assumptions, a rate of convergence can be derived.

The factors that affect the rate of convergence of the empirical risk to the total risk are of interest to the subsequent discussions; therefore, a brief derivation, first given in [2], for the two-category pattern recognition problem with a finite class of functions, is presented here.

Let the total risk of a specific classification function $F(x, \alpha_i)$ be denoted by $P(\alpha_i)$, and the empirical risk of the same be denoted by $v(\alpha_i)$, where α_i is the collection of parameters as defined above. Let (x_1, w_1), (x_2, w_2), $\cdots$, (x_l, w_l) denote the observation-target pairs, and $F(x, \alpha_i), i \in [1, N]$ denote the class of functions used for classification. Then the total risk and the empirical risk can be written as

$$P(\alpha_i) = \int (w - F(x, \alpha_i))^2 P(x, w) dx dw = E_{x,w}[(w - F(x, \alpha_i))^2] \quad (2.4)$$

$$v(\alpha_i) = \frac{1}{l} \sum_{j=1}^{l} (w_j - F(x_j, \alpha))^2 \quad (2.5)$$

As stated above, the law of large numbers dictates that as l approaches

∞, $v(\alpha_i)$ converges to $P(\alpha_i)$. This means that, for all $i \in [1, N]$ and $\varkappa > 0$,

$$\lim_{l \to \infty} Pr(|P(\alpha_i) - v(\alpha_i)| > \varkappa) = 0 \tag{2.6}$$

However, infinite-sample-size convergence alone does not guarantee appropriate behavior in finite-sample-size cases, as demonstrated in the counter-example above. Therefore, to ensure sensible finite-sample-size behavior, a stronger condition would be needed, beyond mere convergence of empirical risk. In other words, for arbitrary $\varkappa > 0$,

$$\lim_{l \to \infty} Pr(\sup_i |P(\alpha_i) - v(\alpha_i)| > \varkappa) = 0 \tag{2.7}$$

This requirement is the requirement of uniform convergence of the empirical risk to the total risk. Note the difference between the two formulations above is that in the case of uniform convergence, the limiting operation does not depend on the α_i used in the evaluation of the risks; i.e., the rates of convergence of all indicator functions within the class of functions $F(x, \alpha_i) \forall i$ conform to the same lower bound.

Define the sequence of events $S_N \colon A_1, \cdots, A_N$ as the classifications made by all the functions within the class $F(x, \alpha_i) \forall i$, and define the possible events to be either "correct classification" where $w_j = F(x_j, \alpha_i)$ or "erroneous classification" where $|w_j - F(x_j, alpha_i)| = 1$. Then the problem of determining convergence properties of the empirical risk to the total risk can be restated in terms of convergence properties of the relative frequencies of events to their probabilities. Observing the counter-example again, and one can see that the category associated with output "0" does not have its frequency converging to its probability; rather, for any finite l, it is infinitely more likely for the classifier to return "0" than any other category. This is an example of nonuniform convergence, and its finite-sample-size behaviors obviously cannot be trusted.

Fortunately, it is easy to establish uniform convergence for a finite class of functions $F(x, \alpha_i) \forall i \in [1, N]$. Since there are N functions within the class, if each of them is evaluated individually on the dataset (x_1, w_1), (x_2, w_2), $\cdots$, (x_l, w_l), then the probability of the largest deviation between P and v being greater than $\varkappa$ is bounded by the sum of the probabilities of

the deviation between each $P(\alpha_i)$ and $v(\alpha_i)$ being greater than $\varkappa$:

$$Pr(\sup_i |P(\alpha_i) - v(\alpha_i)| > \varkappa) \leq \sum_{i=1}^{N} Pr(|P(\alpha_i) - v(\alpha_i)| > \varkappa) \tag{2.8}$$

From the Hoeffding inequality, a specific formulation of the law of large numbers, a bound on the probability of the deviation of v from P for each individual α_i can be derived:

$$Pr(|P(\alpha_i) - v(\alpha_i)| > \varkappa) < 2e^{-2\varkappa^2 l} \tag{2.9}$$

Putting Eqs. (2.8) and (2.9) together, the uniform nature of the convergence of v to P for the class of N functions can be established:

$$Pr(\sup_i |P(\alpha_i) - v(\alpha_i)| > \varkappa) \leq 2Ne^{-2\varkappa^2 l} \tag{2.10}$$

It is easy to see that the above probability tends to 0 as l approaches infinity. Apart from validating the method of empirical risk minimization, uniform convergence provides the additional benefit of allowing the computation of the confidence interval of the true total risk, given the number of possible functions N, and the number of samples l. If η is used to denote $2Ne^{-2\varkappa^2 l}$, then the above equation can be rewritten as

$$Pr(\sup_i |P(\alpha_i) - v(\alpha_i)| > \varkappa) \leq \eta \tag{2.11}$$

And the following three identities can be obtained through the definition of η:

$$\eta = 2Ne^{-2\varkappa^2 l} \tag{2.12}$$

$$\varkappa = \sqrt{\frac{\ln N - \ln(\eta/2)}{2l}} \tag{2.13}$$

$$l = \frac{\ln N - \ln(\eta/2)}{2\varkappa^2} \tag{2.14}$$

Putting Equations (2.11)–(2.14) together, one can say that with confidence of $(1 - \eta)$, the total risk $P(\alpha_i)$ is

$$v(\alpha_i) - \sqrt{\frac{\ln N - \ln(\eta/2)}{2l}} \leq P(\alpha_i) \leq v(\alpha_i) + \sqrt{\frac{\ln N - \ln(\eta/2)}{2l}} \tag{2.15}$$

Let S_f be the set of all functions within the class of functions $F(x, \alpha)$ used in the pattern recognition system. Then N is the cardinality of the set. It is clear from Eq. (2.15) that the confidence interval of $P(\alpha)$ is related to N. As N decreases, and everything else being equal, the confidence interval shrinks. In other words, the minimization of the empirical risk is guaranteed to be a closer approximation to the minimization of the total risk for an S_f with a smaller N value.

However, in this case, it may be more interesting to find a bound on the maximum total risk, instead of a confidence interval of it. Since an appropriate S_f to the specific pattern recognition problem would have its empirical risk close to 0, it is not very informative to have a lower bound that small. Additionally, it is more important for the total risk to be bound from above, so that it never exceeds some value, while allowing it to go as low as it would, since it is bound below by 0.

Also, in favor of a tighter upper bound for $P(\alpha)$, the deviation over all α_i should not all subject to the same bound. That is, for α_i such that $P(\alpha_i)$ is large, more deviation should be tolerated, since $F(x, \alpha_i)$, in this case, is an unfavorable function to use; while for α_i such that $P(\alpha_i)$ is small, the bound on the deviation should be tighter. Therefore, if the bound is then placed on the uniform relative value of the deviation, then

$$Pr(\sup_i \frac{P(\alpha_i) - v(\alpha_i)}{\sqrt{P(\alpha_i)(1 - P(\alpha_i))}} > \varkappa) \tag{2.16}$$

If $P(\alpha_i)$ is small, $\sqrt{P(\alpha_i)(1 - P(\alpha_i))}$ can be approximated by $\sqrt{P(\alpha_i)}$; it is also bound from above by $\frac{1}{2}$. Using McDiarmid's inequality, which is a generalization of Hoeffding's inequality, a bound on the deviation given fixed α_i can be derived:

$$Pr(\frac{P(\alpha_i) - v(\alpha_i)}{\sqrt{P(\alpha_i)}} > \varkappa) < e^{-\frac{1}{2}\varkappa^2 l} \tag{2.17}$$

Then, in the same way in which Eq. (2.9) is generalized to Eq. (2.10), Eq. (2.17) can be generalized to all α_i:

$$Pr(\sup_i \frac{P(\alpha_i) - v(\alpha_i)}{\sqrt{P(\alpha_i)}} > \varkappa) < Ne^{-\frac{1}{2}\varkappa^2 l} \tag{2.18}$$

Let the probability of $P(\alpha_i)$ exceeding its bound $\varkappa$, $Ne^{-\frac{1}{2}\varkappa^2 l}$, be η; then a new set of equations can be derived, demonstrating the relationship between total risk bound $\varkappa$, probability of exceeding the bound η, and the sample size l:

$$\eta = Ne^{-\frac{1}{2}\varkappa^2 l} \tag{2.19}$$

$$\varkappa = \sqrt{2\frac{\ln N - \ln \eta}{l}} \tag{2.20}$$

$$l = 2\frac{\ln N - \ln \eta}{\varkappa^2} \tag{2.21}$$

As done above with the confidence interval, putting Eqs. (2.18)–(2.21) together, then one can say that, for all α_i, with probability $(1 - \eta)$,

$$P(\alpha_i) \leq \frac{\ln N - \ln \eta}{l}(1 + \sqrt{1 + \frac{2v(\alpha_i)l}{\ln N - \ln \eta}}) + v(\alpha_i) \tag{2.22}$$

This result lays the foundations for the rest of the discussion in this thesis, since it asserts that, assuming similar levels of empirical risk v, the smaller N is, the more tightly bounded P is. Recall that N is the number of functions within the class of functions $F(x, \alpha)$. Therefore, this result indicates that, assuming successful empirical evaluation, the smaller the class of functions used to estimate the dependencies, the smaller the total risk. Thus, if the empirical error is kept low, reducing the cardinality of the estimation function set is a valid way to reduce expected error.

2.2 PAC Learning

Similar to the concept presented in the above section is the theory of probably-approximately-correct (PAC) learning, first presented by Valiant in 1984 [8], in the context of machine learning. Within the PAC framework, there is a "learner" who receives randomly drawn training examples, each consisting of an "instance" $x \in X$ and an "outcome" $y \in Y$, where X and Y are the instance and outcome spaces, respectively. The examples are drawn from an underlying joint probability distribution in $X \times Y$ unknown to the learner. Given an instance x in a training example, the learner must choose an "action" a from the decision space A. His goal in choosing his

actions is to minimize a known loss function $l(y, a)$. In achieving this goal, the learner finds a function $h\colon X \mapsto A$ such that its output action, with high probability ("probably"), gives low expected loss ("approximately correct"). If this expected loss becomes arbitrarily low with probability 1, then the task is said to be "learnable." Explorations linking learnability as defined in this framework to the Vapnik-Chervonenkis dimensionality of the function to be learned can be found in [9], [10], and [11]. In the following sections, a version of the derivation found in some of the works listed is presented, modified to be directly applicable to ANNs.

The function h as defined above is called a "decision rule," and is a function chosen from the "decision-rule space" $\mathcal{H}$ (this is analogous to the class of functions $F(x, \alpha)$ used in estimation in the preceding section). Presumably, one or more $h \in \mathcal{H}$ can give the global minimum of expected loss on any given X and Y. If a function $h^*(x)$ is assumed to give the global minimum of expected loss, then the task of the learner is to find a decision rule $h(x)$ such that $h(x)$ is as close to $h^*(x)$ as possible.

Formally, the concept of "as close as possible" can be represented by distances on a metric. In this case, if the expected loss of a decision rule h (the "total risk" in the preceding chapter) is written as $r_h = E_{x,y}[l(h(x), y)]$, then a metric in the decision-rule space can be derived based on the expected losses of the decision rules. Specifically, the distance between two decision rules h_1 and h_2, where $a = r_{h_1}$ and $b = r_{h_2}$, can be written as

$$d_v(a, b) = \frac{|a - b|}{v + a + b} \tag{2.23}$$

where v is any positive real number. Introduced in [12], this measure is similar to the relative quality-measure function $\frac{|a-b|}{b}$, but with several modifications so that it is both well-behaved when a or b is zero-valued and symmetric to the arguments so as to satisfy the requirements of a metric (the reason that a relative measure is used here is the same as in the derivation of the bound on the maximum total risk in the preceding section: a metric relative to the risk values can provide a tighter bound).

2.2.1 The regret function and the big "L" risk

In order to generalize PAC learning and apply it to finding a successful learning algorithm, several more concepts need to be explored. A learning algorithm $\mathcal{A}$, given sample sets consisting of m training examples $\vec{z} = ((x_1, y_1), \cdots, (x_m, y_m))$ drawn randomly from the sample space Z^m according to unknown probability distribution P^m, needs to select, from $\mathcal{H}$, a decision rule $h = \mathcal{A}(\vec{z})$. A "regret" function $L(h, P)$ can be defined on the selection of the decision rule and the underlying probability distribution to measure the closeness of the decision rule produced by $\mathcal{A}$ to the optimal one. The expected value of this regret function, computed over all possible probability distributions, is the definition of the big "L" risk [12].

$$R_{L,\mathcal{A},m}(P) = \int_{\vec{z} \in Z^m} L(P, \mathcal{A}(\vec{z}))dP^m(\vec{z}) \tag{2.24}$$

The big L risk quantitatively assesses the quality of the learning algorithm $\mathcal{A}$. For instance, a specific learning algorithm $\mathcal{A}_1$ may be a 1-hidden-layer, 5-hidden-node feed-forward ANN; to serve as a foil, $\mathcal{A}_2$ is defined to be a 1-hidden-layer, 100-hidden-node feed-forward ANN. Given a specific $\vec{z}$, each ANN can be trained to convergence on $\vec{z}$. For a small m and low-complexity probability distributions P^m, it is likely that the expected losses of the decision rules produced by $\mathcal{A}_2$ are larger on average than the expected losses of those produced by $\mathcal{A}_1$. Then, summing over all possible (low-complexity) $P^m(\vec{z})$, the overall big L risk of $\mathcal{A}_2$ is likely to be higher than that of $\mathcal{A}_1$.

In the PAC framework, a learning algorithm "solves" a learning problem if for all regret functions L and real number $\delta \in (0, 1)$, there exists a finite sample size $m = m(L, \delta)$ such that for all joint distributions P, the big L risk is smaller than or equal to δ. The term $m(L, \delta)$ is then called the "sample complexity" of the learning algorithm. A low sample complexity means that a small big L risk can be guaranteed with a relatively small sample size, which is desirable in dependency-estimation systems.

2.2.2 Sample complexity

It is useful to formulate an expression for bounds on the required sample complexity given δ and a definition of the regret function L. For the

following analysis, the regret function is defined based on the d_v metric presented in Eq. (2.23). For some $v > 0$ and $0 < \alpha < 1$, this specific definition of the regret function L can be written as

$$L_{\alpha,v}(h, P) = \begin{cases} 1 & \text{if } d_v(r_{\hat{h},l}(P), r_{h^*,l}(P)) > \alpha \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

In this case, the big L risk is equal to the probability of the d_v-distance between the optimal decision rule h^* and the decision rule chosen by the learner given m random training examples $\hat{h}$ exceeding α. Given that a learning algorithm solves a learning problem, i.e., $\forall L_{\alpha,v}$ and $0 < \delta < 1$, there exist $R_{L_{\alpha,v},\mathcal{A},m}(P) < \delta$, then $d_v(r_{\hat{h},l}(P), r_{h^*,l}(P)) \leq \alpha$ with probability at least $(1 - \delta)$. However, d_v cannot be computed in this form yet, since the expected loss cannot be determined without knowledge of the underlying probability distribution P. Therefore, it is necessary to estimate the expected loss from the empirical loss. Let $v > 0$, $0 < \alpha$, training samples $\vec{z}$, and $\delta < 1$. If the sample size $m = m(\alpha, v, \delta)$ is such that for all possible probability distributions P, $Pr(\exists h \in \mathcal{H}\colon d_v(\hat{r}_h(\vec{z}), r_h(P)) > \alpha/3) \leq \delta/2$, and the learning algorithm $\mathcal{A}$ is such that $Pr(d_v(\hat{r}_{\mathcal{A}(\vec{z})}, \hat{r}_{h^*}(\vec{z})) > \alpha/3) \leq \delta/2$, then one can say that $Pr(d_v(r_{\mathcal{A}(\vec{z})}(P), r_{h^*}(P)) > \alpha) \leq \delta$. This means that if the learning algorithm $\mathcal{A}$ is able, with certainty $(1 - \delta/2)$, to find a decision rule close to the optimal empirical decision rule for arbitrary $\vec{z}$, and a bound, with certainty $(1 - \delta/2)$, can be developed on d_v between the empirical loss and the expected loss (fortunately, this is similar to the problem solved in the previous section, where a two-sided bound of the total risk based on the empirical risk is developed), then a bound can be given on the big L risk, guaranteeing the solvability of the learning problem by the given algorithm $\mathcal{A}$.

Now, a bound on the sample complexity, dependent on δ, for a given learning algorithm, can be developed. The process of derivation is similar to that presented in Section 2.1.3, where a bound on the probability of the uniform convergence rate of the empirical mean to the expectation is found. In the PAC formulation of the problem, the goal is to find a bound on the sample complexity m such that

$$Pr(\exists h \in \mathcal{H}\colon d_v(E_{xy}[l(y, h(x))], \hat{E}_{\vec{z}}[l(y, h(x))]) > \alpha) < \delta \tag{2.26}$$

The difference here is that instead of the measure used in Section 2.1.3, which is $\frac{E_{xy}[l(y,h(x))]-\hat{E}_{\vec{z}}[l(y,h(x))]}{\sqrt{E_{xy}[l(y,h(x))]}}$, the d_v distance measure, which is a metric, is used. It preserves the tight bounds provided by a relative measure, while giving it convenient metric properties. Using Bernstein's inequality, it can be shown that, if the loss function is bounded between 0 and M [12], then

$$Pr(\exists h \in \mathcal{H} \colon d_v(E_{xy}[l(y,h(x))], \hat{E}_{\vec{z}}[l(y,h(x))]) > \alpha) \leq 2|\mathcal{H}|e^{-\alpha^2 vm/M} \quad (2.27)$$

where $|\mathcal{H}|$ is the size of the decision-rule space. Here a finite decision-rule space is assumed. This result can be generalized to the infinite case by applying a finite ϵ-cover to the infinite space. Now if this probability is desired to be at most δ, then setting $\delta \geq 2|\mathcal{H}|e^{-\alpha^2 vm/M}$ gives a lower bound on the sample complexity m:

$$m \geq \frac{M}{\alpha^2 v}(\ln|\mathcal{H}| + \ln\frac{2}{\delta}) \quad (2.28)$$

Note that the sample complexity decreases with decreasing size of the decision-rule space. This result is analogous to the relationship between the maximum total risk bound and the cardinality of the set of estimation functions obtained in Section 2.1.3; i.e., with a smaller decision-rule space, fewer training examples are needed to achieve the same level of total risk, or a smaller total risk can be achieved with the same number of training examples.

2.2.3 Infinite decision-rule spaces

If discretization due to limitations in computer implementations is discounted, an ANN forms an infinite class of functions with its real-valued parameters. In this case, results similar to those presented above can still be obtained using the concept of ϵ-covers applied to PAC learning as in [12], whose key results are reviewed in this section.

In order to analyze any set T with infinite cardinality, it is necessary to first construct a finite approximation T_0 of the set T, whereas any $t \in T$ can be approximated by a $t_0 \in T_0$ with an error smaller than ϵ. This finite set T_0, therefore, should be a set such that if an ϵ-ball is placed on top of each $t_0 \in T_0$, the infinite set T can be completely covered. If this condition is satisfied, T_0 can be called an "ϵ-cover" of T; the size of the smallest

ϵ-cover of T, given maximum error ϵ, is called the "covering number" of T, denoted by $\mathcal{N}(\epsilon, T)$. For the rest of the discussion, the error ϵ will be defined in terms of the L^1 distance metric.

Consider the infinite class of functions F mapping an input space Z to $[0, M]$ (so the output of any function in F is real and bounded between 0 and M). Let a collection of examples $\vec{z} = (z_1, z_2, \cdots, z_m)$ be randomly generated from Z; applying $\vec{z}$ to an arbitrary function f in F generates an output vector $(f(z_1), f(z_2), \cdots, f(z_m))$. Performing this action to the entire set F, a collection of output vectors, $\{(f(z_1), f(z_2), \cdots, f(z_m))\colon f \in F\}$, can be obtained. This is called the "restriction" of F to $\vec{z}$, and denoted by $F_{|\vec{z}}$. Note that $F_{|\vec{z}}$ is a collection of points within the m-cube $[0, M]^m$. If in the PAC framework considered here, the loss function l is bounded between $[0, M]$, the set F can be immediately taken to represent the set of all possible loss functions $l(h, P)$. Then for each training sample sequence $\vec{z}$ of size m, a restriction of the set of loss functions $F_{|\vec{z}}$ can be obtained, and a covering number $\mathcal{N}(\epsilon, F_{|\vec{z}})$ exists for arbitrary ϵ. The expectation of this covering number over all possible sample sequeces $\vec{z}$ with elements from Z, $E[\mathcal{N}(\epsilon, F_{|\vec{z}})]$, is called the "random covering number" of F. Intuitively, it gives an indication of the "richness" of F on a typical set of m points from Z. This "richness" is analogous to the cardinality of a finite set. Specifically, for finite F, $E[\mathcal{N}(\epsilon, F_{|\vec{z}})] \leq |F|$ $\forall \epsilon$, m, and distributions on Z. Cover numbers can be used in the error-bound derivations of infinite classes of functions in a manner similar to that in which cardinality is used in the error-bound derivations of finite classes of functions, with minor modifications, as follows [12]:

$$Pr(\exists h \in \mathcal{H}\colon d_v(E_{xy}[l(y, h(x))], \hat{E}_{\vec{z}}[l(y, h(x))]) > \alpha) \leq 4E[\mathcal{N}(\alpha v/8, F_{\vec{z}})]e^{-\alpha^2 vm/16M} \quad (2.29)$$

More recent research in [13] has developed better constants on the bound, but they are not important to the demonstration of the relationship among total risk, sample size, and parametric dimensionality, which is the issue to be ultimately addressed by this discussion; therefore, they will not be used here.

Using Eqs. (2.29) and (2.26), a lower bound on sample complexity can

be derived for infinite classes of functions as well:

$$m \geq \frac{16M}{\alpha^2 v}(ln(E[\mathcal{N}(\alpha v/8, F_{\vec{z}})]) + ln(\frac{4}{\delta})) \tag{2.30}$$

All m that satisfy this condition will ensure that the empirical risk obtained on an arbitrary member of the class of decision functions $\mathcal{H}$, both finite and infinite, will deviate from the total risk by more than α with probabilty at most δ. This m, as expressed in Eq. (2.30), is linearly related to the natural logarithm of the random covering number of the class of loss functions. Therefore, intuitively, the more "rich" the class of loss functions, the more samples required to obtain an accurate estimate of the total risk through empirical sampling. To ultimately derive a relationship between parametric dimensionality and total risk in ANNs specifically, a final piece of the puzzle remains: a relationship between parametric dimensionality of an ANN and the random cover number of the class of functions represented by that ANN.

2.2.4 Total loss on infinite decision-rule spaces

In the preceding sections, the relationship between the random cover number of a function-set - a representation of the set's "richness" - and the bounds on the total risk is derived. However, there is as yet no method to determine the random cover number for an arbitrary class of functions. The goal of this section, therefore, is to explain and review results presented in [12], showing conclusively that the random cover number of a class of functions is influenced by the number of adjustable parameters the class of functions possesses, by developing a bound on the random cover number, and relating the bound to the parametric dimensionality of the class of functions.

In order to develop a bound on the random cover number of a set, it is necessary to introduce the concept of "pseudo-dimension." Based on the definition of the Vapnik-Chervonenkis (VC) dimension of a class F of $\{0, 1\}$-valued functions, the concept of pseudo-dimension generalizes the VC dimension to arbitrary classes of real-valued functions.

Before establishing the pseudo-dimension of a set, several definitions need to be presented. Let $sign(x) = 1$ when $x > 0$, 0 when $x \leq 0$, and

let $sign(\vec{x}) = (sign(x_1), \cdots, sign(x_d))$ for some $\vec{x} = (x_1, \cdots, x_d) \in \mathbb{R}^d$. Also, for an arbitrary set $T \in \mathbb{R}^d$, let $sign(T) = \{sign(\vec{x})\colon \vec{x} \in T\}$. Then, for any $\vec{b} = (b_1, \cdots, b_d) \in \{0,1\}^d$, $\{\vec{x} \in \mathbb{R}^d\colon sign(\vec{x}) = \vec{b}\}$ is called the "$\vec{b}$-orthant" of $\mathbb{R}^d$. Hence $sign(T)$, a set of Boolean vectors, indicates the set of orthants intersected by T. For any $T \subset \mathbb{R}^d$, and $\vec{x} \in \mathbb{R}^d$, let $T + \vec{x} = \{\vec{y} + \vec{x}\colon \vec{y} \in T\}$; then, for arbitrary $T \subset \mathbb{R}^d$, if there exists $\vec{x} \in \mathbb{R}^d$ such that $sign(T + \vec{x}) = \{0,1\}^d$, i.e., intersects all orthants of $\mathbb{R}^d$, then T is said to be "full."

Now, let F be a class of functions mapping a set Z into $\mathbb{R}$. For any sequence $\vec{z} = (z_1, \cdots, z_d)$ of elements in Z, let $F_{|\vec{z}} = \{f(z_1), \cdots, f(z_d)\colon f \in F\}$ (the restriction of F to $\vec{z}$). If $F_{|\vec{z}}$ is full, then it is said that $\vec{z}$ is "shattered" by F. The pseudo dimension of F is hence defined to equal to the largest d such that there exists a sequence of d points in Z which is shattered by F, and written as $dim_p(F)$ [12].

A useful result in [14] will be used in the following discussion to bound the random cover number. It is a bound on the size of a finite class of functions F from $S = \{1, 2, \cdots, m\}$ into $\{0,1\}$ with $|F| > 1$. If d is equal to the size of the longest sequence of points $\vec{z}$ from S such that $F_{|\vec{z}} = \{0,1\}^d$, then

$$|F| \leq \sum_{i=0}^{d} \binom{m}{i} \leq (\frac{em}{d})^d \tag{2.31}$$

where e is the base of the natural logarithm.

Another result needed for the derivation is a bound on the "packing numbers" of the set F on a metric d for any $\epsilon > 0$, denoted $\mathcal{M}(\epsilon, F, d)$. For each ϵ, $\mathcal{M}(\epsilon, F, d)$ is the size of the largest ϵ-separable subset G of F under the metric d. In other words, $\mathcal{M}(\epsilon, F, d) = |G|$, where G is the largest subset of F such that $\forall g_0, g_1 \in G, g_0 \neq g_1, d(g_0, g_1) > \epsilon$. The packing numbers of a set are closely related to the covering numbers of the same set. Specifically [12],

$$\mathcal{M}(2\epsilon, F, d) \leq \mathcal{N}(\epsilon, F, d) \leq \mathcal{M}(\epsilon, F, d) \tag{2.32}$$

Intuitively, it is easy to see that, assuming G is 2ϵ-separable, and $|G| = \mathcal{M}(2\epsilon, F, d) > \mathcal{N}(\epsilon, F, d)$, then $\forall g_0, g_1 \in G\colon d(g_0, g_1) > 2\epsilon$, which means that

there is no way that g_0 and g_1 can be covered by the same ϵ-ball. Since this holds for all $g_0, g_1 \in G$, each element in G can only belong to its own ϵ-ball, so G requires $|G|$ ϵ-balls to cover. Its being greater than $\mathcal{N}(\epsilon, F, d)$ contradicts the definition of covering numbers. On the other hand, assuming that G is ϵ-separable, and that $|G| = \mathcal{M}(\epsilon, F, d) < \mathcal{N}(\epsilon, F, d)$, then if an ϵ-ball is placed upon each element in G, together they should emit an ϵ-cover of F. This can be seen if the opposite is assumed. If there exists an element $f \in F \notin G$ outside of the ϵ-cover emitted by placing an ϵ-ball on top of each element in G, then $\hat{G} = G \cup f$ would also be ϵ-separable, and $|\hat{G}| = |G| + 1 > |G| = \mathcal{M}(\epsilon, F, d)$ would produce a contradiction to the definition of packing numbers. Now, if placing an ϵ-ball on top of every element in G produces an ϵ-cover of F, this method requires $|G| < \mathcal{N}(\epsilon, F, d)$ ϵ-balls to cover F, producing a contradiction to the definition of covering numbers. Hence, Eq. (2.32) holds. This means that if a bound on the packing numbers of F can be found, it is concurrently a bound on the covering numbers of F. Therefore, the following discussion tries to establish a bound on the packing numbers of F using the concept of pseudo-dimensionality. From here on, the (pseudo) metric d used on the class of functions F is restricted to the L^1 distance on the probability measure P of the space, denoted by $d_{L^1(P)}$:

$$d_{L^1(P)}(f, g) = E[|f - g|] = \int_Z |f(z) - g(z)| dP(z) \tag{2.33}$$

Take the sign of a random translation of a random restriction of F, $sign(F_{|\vec{z}} - \vec{r})$. This is a function which maps the sequences of elements in Z into $\{0, 1\}$. Therefore, according to Eq. (2.31), the total number of orthants intersected by this function has the constraint

$$|sign(F_{|\vec{z}} - \vec{r})| \leq (em/d)^d \tag{2.34}$$

Let G be a ϵ-separated subset of F, where $|G| = \mathcal{M}(\epsilon, F, d_{L^1(P)})$. Then, $E[|sign(F_{|\vec{z}} - \vec{r})|] \geq E[|sign(G_{|\vec{z}} - \vec{r})|]$ since $G \subset F$. The number of orthants intersected by a random translation of a random restriction of G is greater than or equal to the number of elements in the same translation and restriction of G, which are unique within their respective orthants. In other words, $E[|sign(G_{|\vec{z}} - \vec{r})|] = E[|\{f \in G\colon sign(f_{|\vec{z}} - \vec{r}) \neq sign(g_{|\vec{z}} -$

$\vec{r}) \quad \forall f, g \in G, f \neq g\}]$. This expectation, in turn, can be written as the sum, over all $f \in G$, of the probabilities that f is in its own unique orthant ([12]):

$$\begin{aligned}
E[|sign(F_{|\vec{z}} - \vec{r})|] &\geq E[|sign(G_{|\vec{z}} - \vec{r})|] \\
&\geq E[|\{f \in G\colon sign(f_{|\vec{z}} - \vec{r}) \neq sign(g_{|\vec{z}} - \vec{r}) \\
&\quad \forall f, g \in G, f \neq g\}] \\
&= \sum_{f \in G} Pr(sign(f_{|\vec{z}} - \vec{r}) \neq sign(g_{|\vec{z}} - \vec{r}) \\
&\quad \forall f, g \in G, f \neq g) \\
&= \sum_{f \in G} (1 - Pr(\exists g \in G, g \neq f\colon sign(f_{|\vec{z}} - \vec{r}) = \\
&\quad sign(g_{|\vec{z}} - \vec{r}))) \\
&\geq \sum_{f \in G} (1 - |G| \max_{g \in G, g \neq f} Pr(sign(f_{|\vec{z}} - \vec{r}) = \\
&\quad sign(g_{|\vec{z}} - \vec{r})))
\end{aligned} \tag{2.35}$$

Assuming that all functions in F are bounded by $[0, M]$, $z_i \in \vec{z}$ is drawn at random from probability measure P, and $r_i \in \vec{r}$ is drawn at random from the uniform distribution on $[0, M]$, then if $f, g \in G$, and $f \neq g$, in order for them to be in the same orthant, the random translation $\vec{r}$ cannot lie between them in any dimension, i.e., $\forall i \in [1, m]\colon r_i < \min(f(z_i), g(z_i))$ or $r_i > \max(f(z_i), g(z_i))$, since otherwise the translations would have different signs for the i^{th} dimension, and hence reside in different orthants. Since G is ϵ-separated, $d_{L^1(P)}(f, g) > \epsilon$, and the probability that r_i is between $f(z_i)$ and $g(z_i)$ is at least $\frac{\epsilon}{M}$. Therefore, for all $f, g \in G$, using small number approximations of the exponential function,

$$Pr(sign(f_{|\vec{z}} - \vec{r}) = sign(g_{|\vec{z}} - \vec{r})) \leq (1 - \frac{\epsilon}{M})^m \leq e^{-\epsilon m/M} \tag{2.36}$$

Putting Eqs. (2.35) and (2.36) together,

$$\begin{aligned}
E[|sign(F_{\vec{z}} - \vec{r})|] &\geq |G|(1 - |G|e^{-\epsilon m/M}) \\
&= \mathcal{M}(\epsilon, F, d_{L^1(P)})(1 - \mathcal{M}(\epsilon, F, d_{L^1(P)})e^{-\epsilon m/M})
\end{aligned} \tag{2.37}$$

And using Eqs. (2.34) and (2.37) in conjunction,

$$(\frac{em}{d})^d \geq \mathcal{M}(\epsilon, F, d_{L^1(P)})(1 - \mathcal{M}(\epsilon, F, d_{L^1(P)})e^{-\epsilon m/M}) \tag{2.38}$$

At this point, it is possible to establish the following inequality, summing up the results from above [12]:

$$\mathcal{M}(\epsilon, F, d_{L^1(P)}) < 2(\frac{2eM}{\epsilon} \ln \frac{2eM}{\epsilon})^d \tag{2.39}$$

If $\frac{M}{\epsilon} \ln(2\mathcal{M}(\epsilon, F, d_{L^1(P)})) < d$, it is easy to verify that Eq. (2.39) holds using the fact that $\epsilon \leq M$. Assume that $(M/\epsilon) \ln(2\mathcal{M}(\epsilon, F, d_{L^1(P)})) \geq d$. If the number of samples included in the random restrictions of F performed in the steps above is greater than or equal to $\ln(2\mathcal{M}(\epsilon, F, d_{L^1(P)}))$, then $m \geq d$, and $(1 - \mathcal{M}(\epsilon, F, d_{L^1(P)})e^{-\epsilon m/M}) \geq 1/2$ (the number of samples m can be increased arbitrarily here to suit the needs of the derivation, since it does not factor into the final bound, and is only a construct used in the intermediate steps to facilitate the derivation). Hence, Eq. (2.39) can be obtained from Eq. (2.38), after some algebraic manipulations and substituting in the equivalences.

Equation (2.39) gives a bound on the packing numbers of F in the form of an expression involving the pseudo-dimension of F. Since packing numbers naturally bound covering numbers, it is therefore also a bound on $\mathcal{N}(\epsilon, F, d_{L^1(P)})$. Fortunately, there is a trivial isometry between the (pseudo) metric spaces $(F_{|\vec{z}}, d_{L^1})$ and $(F, d_{L^1(P_{|\vec{z}})})$, where $P_{|\vec{z}}$ is the probability measure based on the empirical observation $\vec{z}$ of Z, where each set has measure equal to the fraction of points it contains in $\vec{z}$. Therefore, the bound in Eq. (2.39) is also a bound on $\mathcal{N}(\epsilon, F_{|\vec{z}}, d_{L^1})$. Substituting this bound into Eq. (2.29), a bound based on the pseudo-dimensionality of the loss function l can be written as

$$Pr(\exists h \in \mathcal{H} \colon d_v(E_{xy}[l(y, h(x))], \hat{E}_{\vec{z}}[l(y, h(x))]) > \alpha) \leq 8(\frac{16eM}{\alpha v} \ln \frac{16eM}{\alpha v})^d e^{-\alpha^2 vm/16M} \tag{2.40}$$

And the sample complexity m, assuming that it is desired for the empirical risk to deviate from the total risk by more than α with probability of at

most δ, is

$$m \geq \frac{16M}{\alpha^2 v}(d \ln(\frac{16eM}{\alpha v} \ln \frac{16eM}{\alpha v}) + \ln \frac{8}{\delta}) \tag{2.41}$$

Note that d in Eqs. (2.40) and (2.41) denotes the pseudo-dimensionality of the loss functions $\{\forall h \in \mathcal{H}\colon l(y, h(x)) \forall x \in X, y \in Y\}$. Assuming the decision-rule space $\mathcal{H}$ is entirely composed of the class of functions F, there is still one additional difficulty: Using the pseudo-dimensionality of the loss functions, instead of that of F, makes the choice of loss functions relevant in the derivation of a general bound on sample complexity. This is a nuisance, since it is more problematic to relate the pseudo-dimensionality of the loss functions to the parametric dimensionality of F. In [12], it was shown that if the loss function is strictly monotonic, then the pseudo-dimensionality of the loss functions remains equal to that of F. This is intuitive, since strictly monotonic functions preserve order. Without loss of generality, consider a strictly increasing loss function l. If $dim_p(F) = d$, then there exists a vector $\vec{x}$ of length d consisted of elements in X and some $\vec{r} \in \mathbb{R}^d$ such that $sign(F_{|\vec{x}} - \vec{r}) = \{0, 1\}$. Now, for any fixed $y \in Y$, the set of all loss functions is $L = \{\forall f \in F, x \in X\colon l(f(x), y)\}$. Denote the restriction of this set on a vector $\vec{x}$ by $L_{|\vec{x}}$. Then for every $\vec{x}$, $sign(F_{|\vec{x}} - \vec{r}) = sign(L_{|\vec{x}} - l(\vec{r}, y))$, due to the strictly increasing l. Therefore, F and L share the same pseudo-dimensionality. Loss functions such as logistic loss and logarithmic loss fall into this category. However, many widely used loss functions do not exhibit strict monotonicity, such as the sum-squared loss function. In that case, it suffices to show that the pseudo-dimensionality of the loss function is at most equal to the pseudo-dimensionality of the decision-rule space. Using the same notation as above, but allowing l to be any well-defined function, and letting $dim_p(L) = d$, it follows that there exists a vector $\vec{x}$ of length d consisting of elements in X and some $\vec{r} \in \mathbb{R}^d$ such that $sign(L_{|\vec{x}} - \vec{r}) = \{0, 1\}$. Therefore, for each $r_i \in \vec{r}$, there exists some f_0 such that $l(f_0(x_i), y) > r_i$, and some f_1 such that $l(f_1(x_i), y) \leq r_i$. Since l is well defined, and $l(f_0(x_i), y) \neq l(f_1(x_i), y)$, $f_0(x_i) \neq f_1(x_i)$. Hence there exists some $r_i' \in \mathbb{R}$ such that it is between $f_0(x_i)$ and $f_1(x_i)$ (possibly overlapping one of them). Without loss of generality, assume $f_0(x_i) > f_1(x_i)$. This means that for all $x_i \in \vec{x}$, there exists some $f_0 \in F$ such that $f_0(x_i) > r_i'$, and some other $f_1 \in F$ such that $f_1(x_i) \leq r_i'$.

Equivalently, for any $\vec{x}$ of size d, $sign(F_{|\vec{x}} - \vec{r'}) = \{0, 1\}$, or $dim_p(F) \geq d$. Therefore, the pseudo-dimensionality of the decision-rule space can be safely used in Eqs. (2.40) and (2.41) without compromising the validity of the bounds.

2.3 ANN-Specific Bounds

In the preceding sections, total risk bounds on both finite and infinite decision-rule spaces have been explored. In the finite case, the bound is affected by the cardinality of the decision-rule space $|F|$; in the infinite case, the pseudo-dimensionality $dim_p(F)$ is used in the bound. However, no explicit connections between either quantity and the parametric dimensionality of an ANN have been established. So far, the discussion has been quite general, in the sense that the results could be applied to any kind of decision-rule space, given that the loss function is bounded. In order to complete this final step, and show that reducing the number of trainable parameters indeed reduces the probability of large deviation of the empirical loss from the total loss, a closer inspection of the ANN architecture is necessary, since the connection between pseudo-dimensionality and parametric dimensionality depends heavily on the specific function structure considered. Based on derivations originally presented in [12], this section presents a somewhat more concise argument on the PAC bounds of the ANN architecture.

ANNs are structures composed of one or more layers of perceptrons, referred to as nodes or neurons in this context. The specific class of ANNs considered in this thesis consists of fully connected feed-forward sigmoidal networks: fully connected acyclic graphs whose vertices are the nodes of the network, and whose edges are the connection weights. Each node has an associated "depth" d, signifying the longest path from said node to any input node at depth 0. "Fully connected" in this context is often taken to mean that every node at depth d is connected to every node at depth $d+1$, but not other nodes. An ANN with connections between nodes whose depths differ by more than 1 is sometimes said to have "shortcuts."

This class of ANNs can be divided again into subclasses according to the difference in the operations performed by each node on its inputs.

For instance, when the node applies a nonlinear function σ to a weighted average of its inputs $\vec{x}$ with bias θ and weights $\vec{w}$, its output $f(\vec{x})$ is equal to $\sigma(\theta + \sum_{j=1}^{n} w_j x_j)$, which is commonly called a quasi-linear unit; if σ operates solely on the square L^2 distance between $\vec{x}$ and some trainable $\vec{a}$, the output becomes $f(\vec{x}) = \sigma(\theta + \sum_{j=1}^{n} (x_j - a_j)^2)$, which is known as a radial basis unit; and if the node transforms its inputs to their natural logarithms before applying the weights, and has a $\sigma'(x) = \sigma(e^x)$, then $f(\vec{x}) = \sigma(e^{\theta} \prod_{j=1}^{n} x_j^{w_j})$, and this is called a product unit. While for the rest of the discussion, most attention will be paid to the analysis of ANNs with only quasi-linear units, the results presented here are general and hold valid over the entire class of fully connected feed-forward ANNs.

For feed-forward ANNs, it is convenient to organize the nodes within the network into layers depending on their depth d. For nodes with the same depth, they are said to be within the same layer. Taking this approach, each layer can be analyzed individually, and the results composed to achieve an overall bound. For an ANN with W trainable parameters and maximum depth d, for each layer $j \in [1, d]$, it can be seen as an independent decision-rule space $\mathcal{H}_j$, accepting the outputs of the previous layers as its inputs. Let the output of any node be bounded between c_0 and c_1. Denote the inputs to the j^{th} layer as X_j. Therefore, any $h_j \in \mathcal{H}_j$ is a function mapping X_j into X_{j+1}. For an ANN without shortcuts, $X_j = [c0, c1]^{n_{j-1}}$, where n_j denotes the number of nodes within the j^{th} layer. For ANNs with shortcuts, let $l_j = \sum_{i=1}^{j} n_i$, then $X_j = [c0, c1]^{l_{j-1}}$. Considering the most general case, it is a vector space of functions taking l_j inputs, and producing n_{j+1} outputs, with its dimension $N = W_j$, the number of trainable parameters within the layer. It is easy to verify that given a class of functions existing within a vector space of dimension W_j, then pseudo-dimension of the class of functions $dim_p(\mathcal{H}_j) \leq W_j$. Therefore, for each layer $\mathcal{N}(\epsilon, \mathcal{H}_j)$ is bounded above by $\mathcal{M}(\epsilon, \mathcal{H}_j)$, which is in turn bounded above by Eq. (2.39), an expression involving $dim_p(\mathcal{H}_j)$ and the number of training samples available M, both known quantities. Since $dim_p(\mathcal{H}_j)$ is bounded above by W_j, it can be used in the inequality without affecting its validity. Now, possessing bounds on all $\mathcal{N}(\epsilon, \mathcal{H}_j)$ in terms of M and W_j, what can be said of the overall decision space $\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_j \times \cdots \times \mathcal{H}_d$?

Assume that for some j, $\mathcal{N}(\epsilon_j, \mathcal{H}_j) = a$. This means that there exists

a set $G \subset \mathcal{H}_j$, $|G| = a$, such that $\forall h \in \mathcal{H}_j$, $\exists g \in G : \ d(h, g) < \epsilon_j$. Now, assume that the Lipschitz bound of the j^{th} layer is b_j. In other words, $\forall f \in \mathcal{H}_j$, $d(\nabla f, 0) \leq b_j$. From this bound, it is clear that for G defined above, $\forall f \in \mathcal{H}_{j+1}$, $\forall h \in \mathcal{H}_j$, $\exists g \in G$:

$$d(f(h), f(g)) < b_{j+1}\epsilon_j \tag{2.42}$$

since the deviation between the function outputs for two different inputs can never be larger than the maximum gradient norm multiplied by the distance between the inputs. A general result encompassing the entire network can then be shown by applying this observation to all the subsequent layers after $\mathcal{H}_j$.

Let $f_i \in \mathcal{H}_{i+1} \times \cdots \times \mathcal{H}_d$, and $G_i \subset \mathcal{H}_i$ such that $|G_i| = \mathcal{N}(\epsilon_i, \mathcal{H}_i)$, and $\forall h_i \in \mathcal{H}_i$, $g_i \in G_i$, $d(h_i, g_i) < \epsilon_i$. Also define $\mathcal{H}^i = \mathcal{H}_1 \times \cdots \times \mathcal{H}_i$, and $G^i \subset \mathcal{H}^i$, $|G^i| = \mathcal{N}(\epsilon_i, \mathcal{H}_i)$, $\forall h \in \mathcal{H}^i$, $g \in G^i$, $d(h, g) > \epsilon_i$. It can be shown that $\mathcal{N}(\sum_{i=1}^{d} [\epsilon \prod_{n=i+1}^{d} b_n], \mathcal{H})$ is at most equal to $\prod_{j=1}^{d} \mathcal{N}(\epsilon, \mathcal{H}_j)$. This can be proven by mathematical induction. As a base case, consider when $d = 1$, and $\mathcal{H} = \mathcal{H}^1$. Clearly, when $d = 1$, $\mathcal{N}(\epsilon, \mathcal{H}^1) = \mathcal{N}(\epsilon, \mathcal{H})$. Thus the base case holds. For the induction step, assume that for $d = k$, $\mathcal{N}(\sum_{i=1}^{d} [\epsilon \prod_{n=i+1}^{d} b_n], \mathcal{H}^k) \leq \prod_{j=1}^{k} \mathcal{N}(\epsilon, \mathcal{H}_j)$. For $d = k + 1$, then, $\mathcal{H}^{k+1} = \mathcal{H}^k \times \mathcal{H}_{k+1}$. Let $\epsilon' = \sum_{i=1}^{d} [\epsilon \prod_{n=i+1}^{d} b_n]$. From the definition of covering numbers and Eq. (2.42), there exists $G^k \subset \mathcal{H}^k$, $|G^k| = \mathcal{N}(\epsilon', \mathcal{H}^k)$: $\forall h \in \mathcal{H}^k$, $g \in G^k$, $h_{k+1} \in \mathcal{H}_{k+1} : \ d(h_{k+1}(h), h_{k+1}(g)) < b_{k+1}\epsilon$. If the covering number of the $(k+1)^{\text{th}}$ layer, $\mathcal{N}(\epsilon, \mathcal{H}_{k+1})$ is known, it means that there exists $G_{k+1} \subset \mathcal{H}_{k+1}$, $|G_{k+1}| = \mathcal{N}(\epsilon, \mathcal{H}_{k+1})$ such that $\forall h_{k+1} \in \mathcal{H}_{k+1}$, $g_{k+1} \in G_{k+1}, d(h_{k+1}, g_{k+1}) < \epsilon$. Therefore, any function $h \in \mathcal{H}^{k+1} = \mathcal{H}^k \times \mathcal{H}_{k+1}$ can be written as the composition of some function $f \in \mathcal{H}^k$ and some function $g \in \mathcal{H}_{k+1}$. Denote h by $g(f)$. For any $f \in \mathcal{H}^k$, $\exists \hat{f} \in G^k$ such that $d(g(\hat{f}), g(f)) < b_{k+1}\epsilon'$ for any $g \in \mathcal{H}_{k+1}$; also, for any $g \in \mathcal{H}_{k+1}$, $\exists \hat{g} \in G_{k+1}$ such that $d(\hat{g}(f), g(f)) < \epsilon$ for any $f \in \mathcal{H}^k$. Since $d(x, y)$ is a metric, the triangle inequality can be used here, and it is easy to see that for any $h = g(f) \in \mathcal{H}^{k+1}$, there exists $\hat{h} = \hat{g}(\hat{f})$ such that $d(\hat{h}, h) < \epsilon + b_{k+1}\epsilon'$. Define the set $G^{k+1} = \{\hat{h} \in \mathcal{H}^{k+1} : \ \hat{h} = \hat{g}(\hat{f}) \ \forall \hat{g} \in G_{k+1}, \ \hat{f} \in G^k\}$. From the preceding inequality, it is obvious that this set emits a cover of the decision-rule space of density $\epsilon + b_{k+1}\epsilon' = \epsilon + b_{k+1} \sum_{i=1}^{k} [\epsilon \prod_{n=i+1}^{k} b_n] = \epsilon + \sum_{i=1}^{k} [\epsilon \prod_{n=i+1}^{k+1} b_n] =$

$\sum_{i=1}^{k+1}[\epsilon\prod_{n=i+1}^{k+1}b_n]$. This shows that for $d=k+1$, $\mathcal{N}(\sum_{i=1}^{d}[\epsilon\prod_{n=i+1}^{d}b_n],\mathcal{H})$ is at most equal to the size of $G^{k+1}=G^k\times G_{k+1}$; so, it is bounded above by $|G^{k+1}|=|G^k||G_{k+1}|\leq\mathcal{N}(\epsilon,\mathcal{H}_{k+1})\prod_{j=1}^{k}\mathcal{N}(\epsilon,\mathcal{H}_j)=\prod_{j=1}^{k+1}\backslash(\epsilon,\mathcal{H}_j)$. Therefore, the inequality also holds for $d=k+1$, and by induction is valid for all $d\in\mathbb{N}$:

$$\mathcal{N}(\sum_{i=1}^{d}[\epsilon\prod_{n=i+1}^{d}b_n],\mathcal{H})\leq\prod_{j=1}^{d}\mathcal{N}(\epsilon,\mathcal{H}_j) \tag{2.43}$$

From Eq. (2.32), for upper bounds, packing numbers $\mathcal{M}(\epsilon,\mathcal{H})$ can be used in place of covering numbers; since an expression has already been developed to bound packing numbers in Eq. (2.39), it can be conveniently used to bound covering numbers as well. Using the parametric dimensionality as an upper bound for the pseudo-dimensionality, for each layer in a feed-forward ANN,

$$\begin{aligned}\mathcal{N}(\epsilon,\mathcal{H}_j)&\leq\mathcal{M}(\epsilon,\mathcal{H}_j)\\&\leq 2(\frac{2e|c_1-c_0|}{\epsilon}\ln(\frac{2e|c_1-c_0|}{\epsilon}))^{W_j}\leq(\frac{2e|c_1-c_0|}{\epsilon})^{2W_j}\end{aligned} \tag{2.44}$$

And overall,

$$\mathcal{N}(\sum_{i=1}^{d}[\epsilon\prod_{n=i+1}^{d}b_n],\mathcal{H})\leq\prod_{j=1}^{d}(\frac{2e|c_1-c_0|}{\epsilon})^{2W_j}\leq(\frac{2e|c_1-c_0|}{\epsilon})^{2W} \tag{2.45}$$

where W is the total number of trainable parameters in the ANN. Substituting Eq. (2.45) into Eq. (2.29), the final deviation of empirical risk from total risk for ANNs can be formulated at last:

$$\begin{aligned}Pr(\exists h\in\mathcal{H}\colon d_v(E_{xy}[l(y,h(x))],\hat{E}_{\vec{z}}[l(y,h(x))])>\alpha)\leq\\4(\frac{c_0 16e|c_1-c_0|d\prod_{n=2}^{d}b_n}{\alpha v})^{2W}e^{-\alpha^2vm/16|c_1-c_0|}\end{aligned} \tag{2.46}$$

Assuming no weight value exceeds β, the maximum number of weights within a layer is $W_{\max}$, and the Lipschitz bounds of the nonlinear functions in any of the nodes do not exceed s, then as derived in [12], the sample

complexity can be written as

$$m = O(\frac{|c_1 - c_0|}{\alpha^2 v}(W(\ln(\frac{c_0|c_1 - c_0|}{\alpha v} + d\ln(s(\beta W_{\max})))) + \ln(\frac{1}{\delta}))) \quad (2.47)$$

where δ is the maximum admissible probability that the empirical risk deviates from the total risk by more than α. This concludes the total risk bound discussions pertaining to ANNs. It can be seen clearly that given the same number of samples, the bound on the probability of large deviations increases exponentially with the number of trainable parameters in an ANN, and the sample complexity, given a maximum probability δ of large deviations between empirical and total risk, increases linearly with the number of trainable parameters. From these relationships, it is clear that if a method to reduce the number of trainable parameters can be employed, without sacrificing the empirical risk, the worst-case total risk can be effectively lowered, i.e., the error-variance of the ANN can be improved, while the bias is not affected. This motivates the subsequent discussions on the new growth methods of ANNs, proposed in Chapter 3.

2.4 Previous Methods

Many researchers have proposed complexity regularization methods in order to achieve a good approximation error/estimation error trade-off. As shown in previous sections, an ANN of arbitrarily large size can approximate any function, but the deviation between the empirical risk and the total risk when the ANN is only trained on a finite amount of data can also grow arbitrarily large. Therefore, it is important to limit the number of trainable parameters in an ANN to avoid overfitting, or learning the "noise" in a finite dataset.

2.4.1 Complexity minimization

Barron [3], [4] and Barron and Cover [7] proposed a method of complexity minimization based on the minimum description principle as described in [15], applied to feed-forward ANNs. According to the minimum description length principle, the minimum complexity decision rule is defined as the

decision rule $\hat{h} \in \mathcal{H}$ such that

$$\hat{h} = \min_{q}(\mathcal{L}(q) + H(l_q(\vec{z}))) \tag{2.48}$$

where $l_q(\vec{z})$ is the loss function evaluated when the decision rule q is applied to any sequence of testing samples $\vec{z} = \{z_1, \ldots, z_i, \ldots, z_m\}$, $z_i = (x_i, y_i)$, for $x_i \in X$, the input space, and $y_i \in Y$, the target space; $H(x)$ denotes the entropy of x; and $\mathcal{L}(q)$ denotes the description length of the decision rule q itself. This minimization penalizes overly complex decision rules and favors simple decision rules that substantially reduce entropy. It is shown in [7] that if the relationship between X and Y is deterministic and in the decision-rule space $\mathcal{H}$, then for sufficiently large sample sizes, it is discovered by $\hat{h}$ with probability 1 (the loss function for $\langle$ goes to 0 as m approaches infinity). However, with smaller numbers of m, one cannot guarantee that the true relationship is discovered in the same manner. In [7], it is shown that the squared Hellinger distance between the minimum description decision rule given m samples, $\hat{h}_m$, and the zero-loss decision rule, h^*, defined as $d_H^2(h^*, \hat{h}_m) = \int (\sqrt{h^*} - \sqrt{\hat{h}_m})^2$, is bounded in order-of-magnitude by $R_m(h^*)$, the index of resolvability of $h*$, defined as

$$R_m(h^*) = \min_{q}(\frac{\mathcal{L}(q)}{m + D(h^* \| q)}) \tag{2.49}$$

where $D(h^* \| q)$ is the relative entropy between h^* and q. The bound, then, is expressed as

$$d_H^2(h^*, \hat{h}_m) \leq O(R_m(h^*)) \tag{2.50}$$

Therefore, the index of resolvability can be used to bound the deviation between empirical risk and expected risk. In [4], the author proposed a bound, in probability, of the total loss without depending on knowledge of the empirical loss, but through an analysis of the "density" of the class of ANNs with n nodes on a bounded output space. It is shown that for decision rules implemented by a three-layer ANN with n nodes, h_n, the squared-distance between h^* and h_n^*, where h_n^* is defined as the best decision rule among all h_n, is bounded by $\frac{C_{h^*}}{\sqrt{n}}$, where C_{h^*} is the first absolute moment of the Fourier magnitude distribution of h^*. The total error, then,

is bounded by the sum of the estimation error, defined as the difference between the best empirical decision rule $\hat{h}_n$ and the best decision rule h_n^* within the class of decision rules implemented by three-layer ANNs with n nodes, and the approximation error, defined as the difference between the best decision rule h_n^* among all h_n, and the true relationship between the input space X and the target space Y, h^*. A method to select the optimal network architecture is also proposed based on these bounds, simultaneously minimizing the approximation error as well as the estimation error. These bounds and the complexity-minimization method were extended to four-layer ANNs in [16].

The main disadvantage of this method is that, for specific problems, the complexity-minimization criterion can be difficult to formulate, and the minimization can be problematic to perform. The function C_{h^*} is most often unknown beforehand, so in order for a complexity-minimization-based decision-rule to fulfill this optimality criterion, additional assumptions need to be made about the a priori joint probability distribution of the input and target spaces. However, despite these complications in implementation, it is a nice demonstration of the finite-sample convergence properties of ANNs, showing, in an alternative way to the derivations presented in the preceding sections, the relationship between the number of nodes in a network and the probability of error, conveniently summarizing the approximation error/estimation error trade-off (bias/variance trade-off) in one bound expression.

2.4.2 Other notable methods

There are a number of other notable methods for determining good structures of ANNs, given a training dataset. These methods are either less generally applicable, or less mathematically rigorous, than the methods discussed in previous sections, but have attractive properties such as ease of practical implementation and low computational complexity. Brief overviews of these methods are given in the following sections, with the intention of highlighting their strengths and limitations.

Cross-Validation

The method of cross-validation, very widely employed in practice due to its relative simplicity and good performance, is a simple method only capable of indicating whether "overfitting" has occurred. This method can be used either to stop training before overfitting occurs, or to find a good number of nodes.

Before training begins, a fraction of the training data is withheld from the ANN to be trained. These data points are reserved as cross-validation data; after each epoch of training, the empirical error of the cross-validation set is evaluated. Early on, this error will decrease along with the empirical error of the training dataset; however, if overfitting of the training dataset begins to become pronounced, the empirical error of the cross-validation set will begin to increase with each additional iteration. As soon as this is observed, training is stopped to avoid overfitting [17].

As an alternative method, the ANN can be allowed to be trained to convergence, and then the cross-validation is performed. This process can be repeated for ANNs with different numbers of nodes, and the best performer is selected.

Clearly, this method needs a large training dataset to work properly. To have good confidence that the empirical error evaluated on the validation set reflects the performance of the ANN on other test data, the validation set needs to be relatively large, but that detracts from the size of the training dataset. So for applications where data-gathering is difficult, this method cannot be easily employed.

Pruning and the Optimal Brain Surgeon

Another way of ensuring that the ANN does not have more parameters than it needs to approximate the function is to train an ANN with enough parameters, then "prune" away the parameters that contribute little to the overall error rate. Among various pruning methods, the most simple-minded approach is pruning by magnitude of the parameter: At each pruning iteration, the parameter with the smallest magnitude is selected, and if its magnitude is small enough, e.g., smaller than some threshold, it is pruned away from the ANN. The network is then retrained, and this

process is repeated until the parameter with the smallest magnitude in the ANN does not satisfy the pruning criterion, at which point pruning stops [17].

This is a simple and easy approach, but it assumes that parameters with smaller magnitudes contribute less to decreasing the error rate. This assumption does not hold if the partial derivative and the second-order partial derivative of the loss function with respect to a parameter are very large; even if the parameter has a small magnitude, the error will be substantially increased with its removal. The optimal brain surgeon (OBS) proposed in [18], on the other hand, tries to take into account the gradient and the Hessian of the error with respect to the parameters of the ANN. Consider a trained ANN, whose parameters are at a local minimum of the error surface: Assuming that close to the local minimum, the error surface can be well-enough approximated by a second-order function of the parameters, then given the gradient and Hessian of the parameter vector, one can compute approximately the increase in error as a function of the change in parameters. Therefore, the parameter whose elimination would cause the slightest increase in error is pruned from the ANN. In [18], the author also described the process by which one can adjust the remaining parameters in the ANN after each iteration of pruning so that, assuming the error surface is approximately second-order around the local minimum, gradient and Hessian information can be used to obtain the new values for the remaining parameters without retraining the whole ANN, thus saving pruning time. Therefore, when an OBS operates on a trained ANN, the parameter whose elimination influences the error in the slightest way is pruned, and then the remaining parameters are adjusted to keep them at the local minimum of the error surface. This process is repeated until no more parameters can be pruned, at which point the ANN is retrained on the training dataset.

OBS is relatively easy to implement; while the required inverse Hessian can be computationally inhibitive, there are several approximate recursive methods to expedite its computation.

Entropic Optimum ANN Synthesis

Introduced in [19], the method of entropic optimum ANN synthesis is only applicable to classification ANNs with the step function as the nonlinear function within each node. By considering the entropy of the inputs and the targets, it can be seen that for each output class, for all inputs p_i belonging to that class, $H_{\text{in}} = \sum_i (p_i \log p_i) \geq (\sum_i p_i) \log(\sum_i p_i) = H_{\text{out}}$. Therefore, each layer in the ANN can be seen as an information filter, keeping the useful information and rejecting the rest. Assuming that there is no ambiguous input sample (belonging to more than one output class), the loss of useful information $\Delta = H_{\text{out}} - H^*_{\text{out}}$, where H_{out} is the entropy of the output, and H^*_{out} denotes the entropy computed at the output of the ANN. Assuming no ambiguity, the synthesis process is as follows:

1. Initialize the ANN to input layer only.
2. Create a new layer with no node.
3. Create a new neuron, and use some optimization algorithm to find the parameters of the node giving minimum Δ.
4. If $\Delta = 0$, then layer synthesis ends. Otherwise, repeat last step.
5. If the number of output classes of the current layer is equal to the number of target classes, synthesis ends. Otherwise, go to step 2.

This method has the advantage that only one node is updated at any given time, thus increasing training speed substantially. However, it is only applicable to step function nonlinearities, which renders it useless for regression applications. Also, it is optimal in a greedy sense; at the creation of every node, it tries to minimize Δ. This does not lead to the identification of the optimal set of nodes and parameters which work together to achieve zero information loss.

Cascade-Correlation ANN Synthesis

Another method proposed to incrementally synthesize an ANN, the cascade-correlation network architecture, differs from the construction of common layered ANNs [20]. The network starts out with no hidden node; each output has weighted connections to each input, with adjustable

weights. Then one or several candidate nodes are created, possibly with different nonlinearities; they have weighted connections to all the inputs, as well as the outputs. The input weights to the newly created nodes are adjusted to maximize the correlation of the node output and the residual error, using an iterative update optimization algorithm. The best of all candidate nodes is chosen to be kept, its input-side weights are frozen, and the other candidate nodes are thrown away. This process is repeated, with each new node receiving weighted connections from all inputs and previously selected nodes, and each output has weighted connections to all inputs and nodes, until the decrease in error becomes stagnant.

This method creates a feed-forward ANN with multiple hidden layers, with only one node in each hidden layer, having shortcut connections to all the previous layers and inputs. The iterative synthesis process is demonstrated in Fig. 2.1.

The advantage of this method is similar to the entropic optimum ANN synthesis method: the nodes are created one after another, so training is very fast, since the number of parameters adjusted at each time is small. It is capable of automatically finding the smallest number of nodes needed to achieve some error goal. The cascade-correlation ANNs have the additional benefit of accommodating heterogeneous nonlinearities in their nodes, making solutions to problems with complex class boundaries possibly more elegant. However, it also suffers from a greedy optimization; it does not discover a globally optimal structure.

Apart from the methods mentioned above, several other approaches, including training ANN ensembles incrementally and combining them with boosting [21], [22], and testing the generalizability of the ANN by introducing noise in the training dataset [23], have also been proposed.

2.5 Motivation for a Growth Method

All the complexity regularization methods discussed in the previous section aim to achieve a good bias/variance trade-off by minimizing the number of parameters in an ANN, thus diminishing the error variance, while keeping an acceptable level of error bias. In a sense, one loses with all of these approaches. Given a small and noisy dataset, the best one can do is

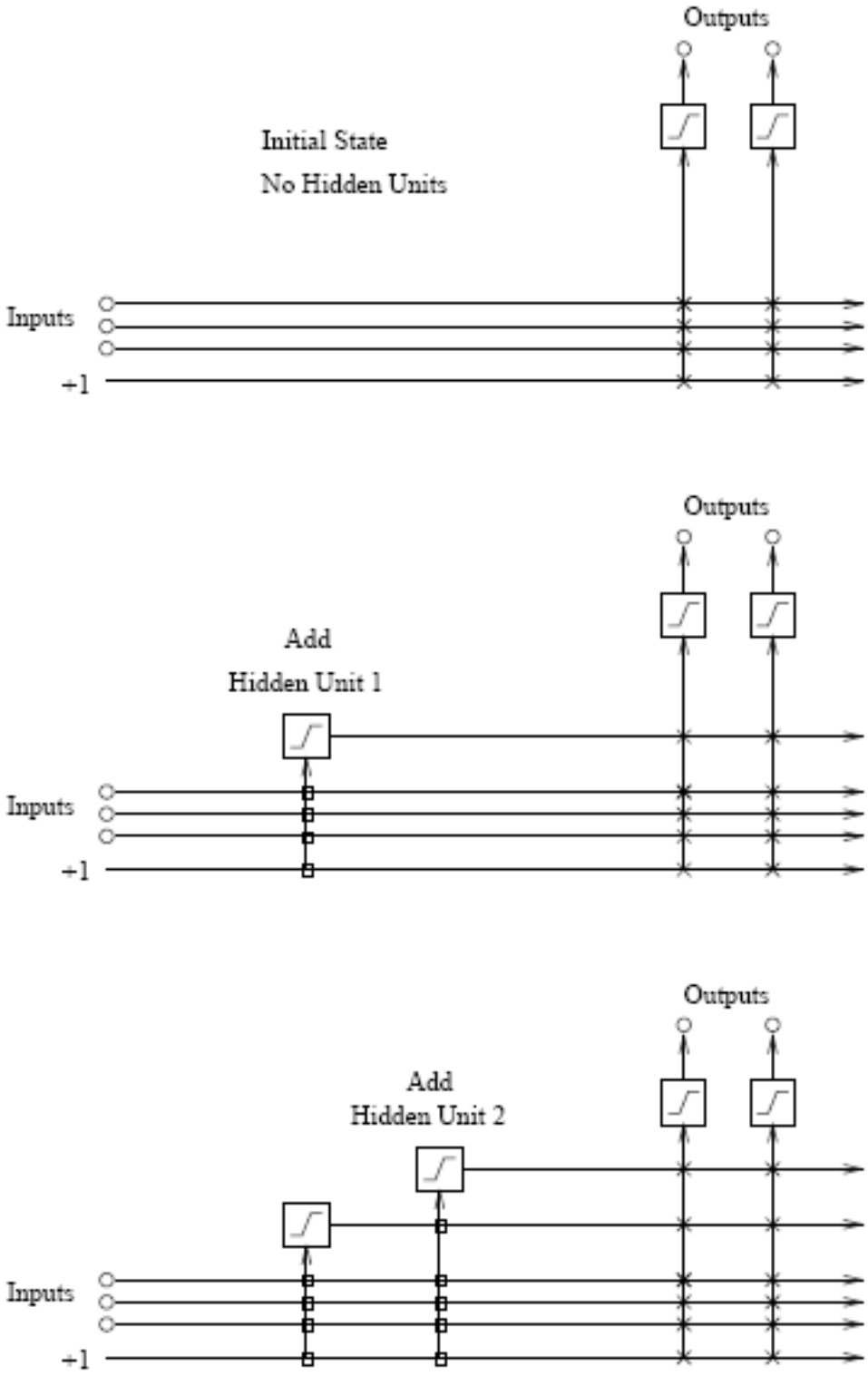

Figure 2.1: Cascade-correlation ANN synthesis. Square connections are only adjusted when the node is inserted. Cross connections are always adjustable.

to severely reduce the number of parameters in an ANN, and be content with a possibly large overall error bias. Of course, the precise underlying relationship between the inputs and targets can never be divined solely from small and noisy datasets. However, if there exists a large and accurate dataset whose inputs and targets share a relationship similar to that within the small and noisy dataset, then presumably there is something to be gained through studying the large and accurate dataset and understanding its input-target relationship. In order for it to be helpful, the knowledge gained here needs to be put into a form that can be readily incorporated into ANN training. With this goal in mind, Chapter 3 describes the method by which one can "grow" an ANN trained on large and accurate datasets, transforming it to approximate a new function in the process. The growth typically requires fewer parameters to be trained than creating and training a new ANN, decreasing the error variance; and since the growth method does not affect the number of parameters in the original ANN, error bias can also be preserved, if the applications are similar enough. In other words, by incorporating knowledge from a similar application in the form of a well-trained ANN, the performance and generalizability of the ANN for the problem at hand can be improved simultaneously, diminishing the effects of the bias/variance trade-off to some degree.

3 ANN GROWTH AND TRAINING

From the explorations in the previous chapter, it is clear that the generalizability of an estimator or decision rule is dependent on the number of trainable parameters within the estimator, as well as the number of available training samples. If an application has strict restrictions on the number of training samples due to data collection difficulties, then reducing the number of parameters becomes the only viable option in order to improve generalizability. All of the methods discussed in the preceding chapter share the dilemma that there is a performance/generalizability trade-off manifested in the relationship between the number of trainable parameters in an ANN and the approximating power of said ANN. The fewer trainable parameters an ANN possesses, the smaller the decision-rule space it is capable of covering, and thus the less its approximating power. However, the more trainable parameters an ANN possesses, the more probable that the empirical error will have large deviations from the true total error, and thus the less generalizable the ANN becomes. There is no way to circumvent this trade-off without increasing the size-requirement of the training dataset.

Let the instance space of interest here be X, and the outcome space be Y. In order to train an estimator such that $d(h^*(x), y)$ is minimized for any instance-outcome pair $z = (x, y)$, a training dataset $\vec{z}$ is collected, so that an empirical-error minimization method can be applied to the ANN to minimize its error over $\vec{z}$. If the size of $\vec{z}$ is small, in order to ensure the minimization of error over $\vec{z}$ leads to minimization of error over $Z = X \times Y$, the decision-rule space $\mathcal{H}_{\text{sm}}$ can only contain ANNs with a small number of trainable parameters, so the best estimator $h^*_{\text{sm}} \in \mathcal{H}_{\text{sm}}$ may not approximate h^* well if H^* is complex. However, assume that there exists another instance space X' and another outcome space Y' with a joint probability distribution related to that of X and Y, and a $\vec{z}'$ whose size is much larger than that of $\vec{z}$; here a better performance/generalizability trade-off is real-

izable due to the larger amount of training data, so an $h'_{\text{lg}} \in \mathcal{H}'_{\text{lg}}$ is found using some empirical-error minimization method with good performance and generalizability. If then all the parameters in h'_{lg} have their values frozen, and a relatively small growth occurs on h'_{lg}, causing it to acquire more nodes/layers, then the grown ANN will implement a decision-rule space $\mathcal{H}_{\text{gr}}$. The "grown" nodes are responsible for adapting the functional structure of the decision rules implemented by the original ANN, trained on $\vec{z}'$, to approximate the relationship between X and Y. Due to the similarity between the joint disctributions of X, Y and X', Y', only a small number of parameters is required for the adaptation, improving the generalizability of the grown ANN, while benefiting from the performance of the original ANN, since the nodes with frozen parameters implement approximations of function-structural elements between X' and Y', which are simply adapted by the grown nodes to approximate similar function-structural elements between X and Y.

The following discussion will clarify what it means to "grow" a network, what kinds of "adaptation" can occur when a network is grown, and what it means for two joint distributions to be "related" in this context.

3.1 Growing an ANN

Consider the case where an ANN is trained to approximate an unknown system implementing a function f from the instance space X to the outcome space Y (Fig. 3.1). Three-layer ANNs with squashing nonlinearities are universal approximators: with indefinitely many hidden nodes, an ANN can approximate any deterministic measurable function with arbitrarily small errors ([1]). However, with a limited number of hidden nodes, the approximation accuracy is thus limited as well. If for an ANN with n hidden nodes, the error criterion on a training dataset cannot be met, one must increase the number of hidden nodes to diminish the training error. The following sections introduce two different approaches to growing ANNs, and analyze their effects, for simplicity's sake, on an ANN with step-function nonlinearities.

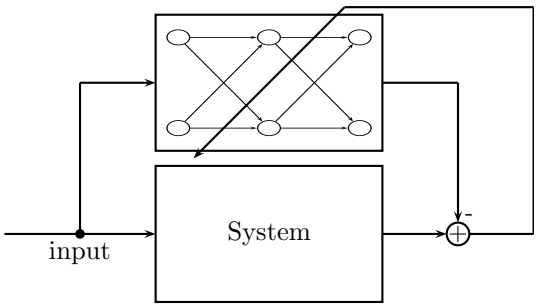

Figure 3.1: ANN approximating a system.

3.1.1 Increasing the breadth of an ANN

When a three-layer step-function ANN is trained on a training dataset, it divides the input space into Voronoi regions, with each region assigned an output. Assuming the weight-vectors of the hidden nodes are linearly independent, the number of Voronoi boundaries is equal to n, where n is the number of hidden nodes in the first hidden layer of the ANN. This is due to the fact that each hidden step-function node effectively outputs a binary number in $\{-1, 1\}$ depending on the sign of the result of the expression $\vec{w}^T\vec{x} + b$, where $\vec{w}$ is the weight-vector of the node, $\vec{x}$ is an input sample, and b is the bias of the node. The equation $\vec{w}^T\vec{x} + b = 0$ is a hyperplane in the input space X, with $\vec{w}$ as a normal vector, forming a Voronoi boundary. Therefore, each combination of hidden-layer outputs corresponds with a Voronoi region with such linear boundaries. When one increases the number of hidden nodes in the first hidden layer of an ANN, he increases the number of Voronoi boundaries, making finer divisions of the input space. For example, for the ANN in Fig. 3.1, let $\vec{w}_1 = [\,1\ 1\,]^T$, $\vec{w}_2 = [\,1\ {-1}\,]^T$, $b_1 = b_2 = 0$, then the Voronoi boundaries would be as shown in Fig. 3.2. One way to grow this ANN would be to increase its breadth. In this context, it means the addition of untrained nodes to the first hidden layer of the ANN, as demonstrated in Fig. 3.3. This operation has interesting consequences. Since the previously trained nodes are not reinitialized, the existent Voronoi boundaries are not disturbed. On the other hand, a new node is added in the demonstration above; consequently, the number of Voronoi boundaries of the input space is increased by one, which means that some previous intact Voronoi regions have a new boundary added to their interiors, dividing each of them further into two disjoint regions.

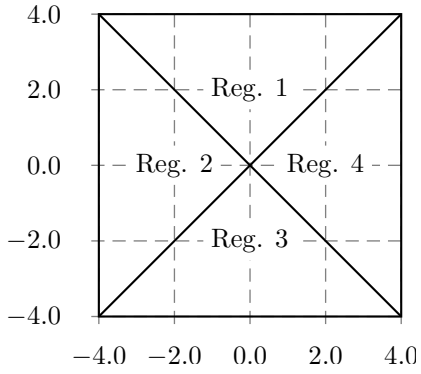

Figure 3.2: Voronoi boundaries of an ANN.

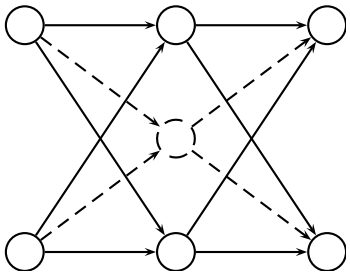

Figure 3.3: Increasing the breadth of an ANN.

The position of this new boundary is determined by the bias and weights of the added node, along the hyperplane defined by $\vec{w}^T\vec{x} = b$. Figure 3.4 shows the effects of adding one hidden node to the ANN shown in Fig. 3.1, with the same parameters for the nodes present in Fig. 3.3, and assuming $\vec{w}_3 = [\,0\ 1\,]^T$, and $b_3 = -1$.

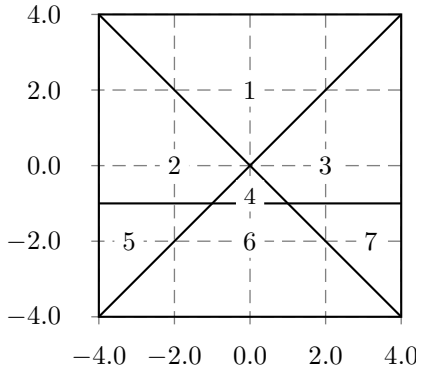

Figure 3.4: Voronoi boundaries after increasing the breadth of an ANN.

3.1.2 Increasing the depth of an ANN

Increasing the depth of an ANN, in this context, involves more steps to make it analytically attractive. A naïve approach to adding an additional layer to the network is to make the output layer of the original ANN a hidden layer instead, and create a new output layer. But this approach effectively eliminates all the Voronoi boundaries developed in the original training, because the output only depends on the second hidden layer, which has as many nodes as the dimensionality of the original output; the hyperplanes implemented by the added layer can only overlap the original ANN's classification boundaries, regardless of Voronoi boundaries implemented by the first layer of the original ANN. Thus, increasing the depth of an ANN in this way only maps each original output to some new output (category-remapping), which is of limited utility. For instance, if the ANN in Fig. 3.3 is used as the basis for extension, the resultant ANN is shown in Fig. 3.5. Let $\vec{w}_{j,k}$ denote the weight-vector of the k^{th} node in the j^{th} layer, let $b_{j,k}$ denote the bias at the same node, and assume $\vec{w}_{3,1} = [\,0.5\ 0\ 0\,]^T$, $\vec{w}_{3,2} = [\,0\ 0.5\ 0\,]^T$, $\vec{w}_{4,1} = [\,-1\ 0\,]^T$, $\vec{w}_{4,2} = [\,0\ -1\,]^T$, $b_{3,1} = b_{3,2} = b_{4,1} = b_{4,2} = 0$; then the output assignment for the Voronoi regions

for the original ANN is shown in Fig. 3.6, and the new output assignment is shown in Fig. 3.7.

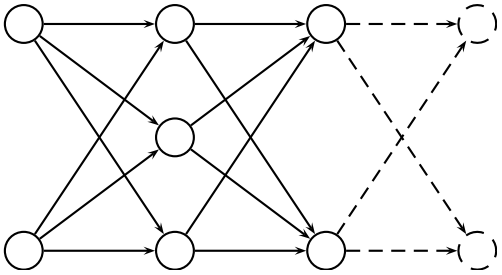

Figure 3.5: Increasing the depth of an ANN naïvely.

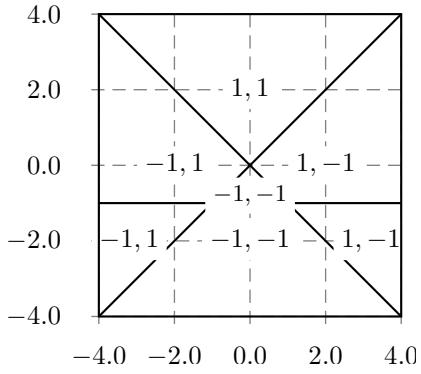

Figure 3.6: Output assignments of an ANN.

It is possible to increase the depth of an ANN without resulting in a category-remapping seen in Fig. 3.7. As is often the case, the outputs of the nodes in the first hidden layer divide the input space and separate various "features" of the input from each other; the output layer integrates the salience of each feature, and categorizes the input instance. It is of greater utility if the increase in depth of an ANN results in a feature-remapping of the original ANN. This can be achieved with the following steps:

1. Make the original output layer a linear layer.

2. Create one or more new hidden nodes in the same layer as the original output node.

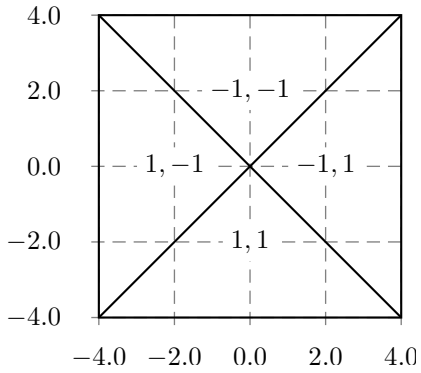

Figure 3.7: Output assignments after naïve increase of depth.

3. Create an additional output layer with the same number of nodes as the original output layer. Each i^{th} node in the new output layer only has trainable weighted connections with the newly created node(s) in the previous step; additionally, each i^{th} node shares a fixed unity-gain connection with the i^{th} node in the original output layer, and is not connected to any other node.

Applying this approach to the same base ANN as the example yields Fig. 3.8.

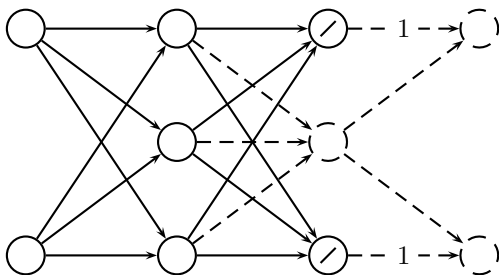

Figure 3.8: Increasing the depth of an ANN.

This approach does not obliterate the Voronoi boundaries from the first layer of the original ANN. The function implemented by the original ANN can be written as

$$\vec{y}_{\text{old}} = \vec{\text{step}}(W_2^T \vec{\text{step}}(W_1^T \vec{x} + \vec{b}_1) + \vec{b}_2) \tag{3.1}$$

where $W_j = [\,\vec{w}_{j,1}\ \vec{w}_{j,2}\ \cdots\,]$ is the matrix composed of the weight-vectors of all the nodes within the j^{th} layer. Since the original output layer has been

made linear, the function implemented by the first two layers now becomes

$$\vec{y}_{\text{lin}} = W_2^T \vec{\text{step}}(W_1^T \vec{x} + \vec{b}_1) + \vec{b}_2 \tag{3.2}$$

And at the new output layer, the function implemented is

$$\vec{y} = \vec{\text{step}}\left(\begin{bmatrix} I_K \\ W_{3,new} \end{bmatrix}^T \vec{h} + \vec{b}_3 \right) \tag{3.3}$$

where:

- $\vec{h} = \left[\vec{y}_{\text{lin}}^T \ \vec{\text{step}}(W_{2,\text{new}}^T \vec{\text{step}}(W_1^T \vec{x} + \vec{b_1}) + \vec{b}_{2,\text{new}})^T \right]^T$.
- I_K is an identity-matrix of K rows and columns, where K is the number of original outputs.
- $W_{2,\text{new}}$ and $\vec{b}_{2,\text{new}}$ are the adaptable weight-vector(s) and bias(es) belonging to the node(s) added to the original output layer.
- $W_{3,\text{new}}$ is the matrix composed of the adaptable weight-vectors of each node in the newly created output layer (the only adaptable weights are the ones associated with the newly-created nodes in the original output layer). $\vec{b}_3$ is the bias-vector of the new output layer.

It is evident that $\vec{h}$ is directly related to the original Voronoi regions created by the first layer, since all elements of $\vec{h}$ are functions of $\vec{\text{step}}(W_1^T \vec{x} + \vec{b}_1)$. Namely, for every input sample within the same original Voronoi region, $\vec{h}$ will result in the same value. Since the inner product of $\vec{w}_{3,\text{new},n}^T \vec{\text{step}}(W_{2,new}^T \vec{\text{step}}(W_1^T \vec{x} + \vec{b_1}) + \vec{b}_{2,\text{new}}) + b_{3,n}$ is added to each n^{th} output node, the new output nodes are still functions of the features of the instance space, rather than becoming functions of only the category-assignments of the original ANN. Each new output becomes a modified version of previous output assigned to each Voronoi region specified by the first layer due to the new hidden layer. Each additional node in the new hidden layer further divides the Voronoi regions created from the first hidden layer into two groups by erecting a hyperplane in the output space of the first hidden layer; depending on which side of the hyperplane the Voronoi-region assignment of a specific input instance falls, either a positive or a negative additive modification is effected by that node in the new hidden layer; the

effect the additive modification has on the n^{th} node in the new output layer is dependent on the weight-value between the node effecting the modification and the n^{th} output node. As an example, assuming $\vec{w}_{2,3} = \begin{bmatrix} 1 & -3 & 1 \end{bmatrix}$, $w_{3,1} = 0.4$, $w_{3,2} = -0.6$, $b_{2,3} = b_{3,1} = b_{3,2} = 0$, then in this case, the output assignments are shown in Fig. 3.9. Compare Fig. 3.7 and Fig. 3.9.

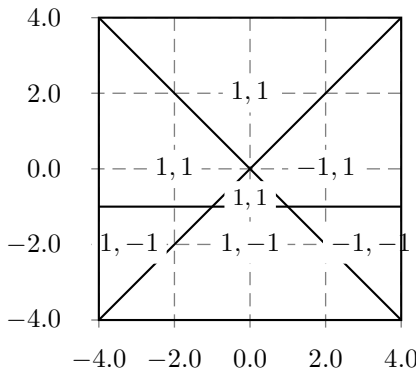

Figure 3.9: Output assignments after increase of depth.

There is an obvious difference in that the latter depth-growth method, as analyzed above, is capable of making the output-category modifications a function of the Voronoi-region assignment in the first layer of an input instance, instead of a function of the categorization by the original ANN of the input instance.

The composition of ANN growth methods in both breadth and depth allows one to "fine-tune" the existent Voronoi regions of a step-function ANN. By increasing the breadth of the ANN, one can add additional Voronoi boundaries, thus making finer divisions of the input space; and by increasing the depth of the ANN, one can additively change the output value assigned to each region without fundamentally altering the structure. This property of said methods allows this kind of extension to be applied to existing ANNs to achieve feature redivision or remapping without training entirely new ANNs.

3.1.3 Arbitrary nonlinearities

In practice, few ANNs use the step-function nonlinearity due to the discontinuous error-surface caused by the discontinuous behavior of the step function, since most iterative optimization algorithms depend on the con-

tinuous differentiability of the loss function. The effects of using nonlinearities other than the step function are explored in a qualitative way in the following sections. The problems of pattern classification and function regression will be considered separately.

Classification

ANNs constructed for classification usually use a hypertangent function as the nonlinearity in their nodes, since it approximates the step function in the sense that its output approaches positive or negative 1 as its input approaches positive or negative infinity, respectively, and crosses zero at the origin. If only the signs of the node-outputs are considered, the hypertangent function is equivalent to the step function. In principle, the effects of ANN growth are not significantly altered by using hypertangent functions instead of step functions. However, the fact that the output of a hypertangent function can be any real number in $(-1, 1)$ means that the output space of each layer, instead of only consisting of 2^k points where k is the number of nodes within that layer, is now $(-1, 1)^k$. Therefore, each breadth-growth made increases the dimensionality of the layer's output space by 1, and a depth-growth creates a hyperplane in that output space, and the feature-remapping that occurs along with the depth-growth is now not just a function of the Voronoi-region assignments made in the first layer; instead of being a function only of the signs of the node-outputs from the first layer, it is also a function of the distances between the input instance and hyperplanes formed by the nodes in the first layer. When the input instance is sufficiently far from all the hyperplanes, the hypertangent function approximates the step function; however, if another input instance lies in the same Voronoi region as the first, but is close to some hyperplane, it could be feature-remapped to a different category.

Regression

For regression ANNs, the situation becomes somewhat more diverse. Since the goal of the network is not to classify, the paradigm adopted in analyzing the ANN through the Voronoi regions it imposes on the input space is not of great utility here. If other nonlinearities are also considered — for instance, polynomial functions — the step-function approximation breaks

down completely, so it is necessary to analyze the effects in terms other than Voronoi regions.

Let the function implemented by the first layer of the original ANN be called $\vec{f_1} = [\, f_{1,1} \cdots f_{n_1,1} \,]$, which maps the input space to $\mathbb{R}^{n_1}$, where n_1 is the number of nodes within the first layer. Each node in the first layer implements a function $f_{i,1}$ both of the input's halfspace assignment made by the hyperplane formed by its weight-vector and bias, and of the input's distance from the same hyperplane. A breadth-growth in this layer would result in an increase in the dimensionality of $\vec{f_1}$, while a depth-growth would mean the inclusion of an additive modification to the regression results as a function of the outputs of this layer. Assuming the output layer is composed of linear nodes: If the vector-function implemented by the original ANN's hidden layer is denoted as $\vec{f_1}$, the vector-function implemented by the breadth-growth(s) to the original ANN as $\vec{f_1'}$, and the vector-function implemented by the depth-growth as $\vec{f_2'}$, then the new ANN implements the function

$$\vec{y} = \begin{bmatrix} I_K \\ W_{3,new} \end{bmatrix}^T \vec{h}' + \vec{b}_3 \tag{3.4}$$

where $\vec{h}'$ is defined as $[\, \vec{y}_{\text{lin}}^T \;\; \vec{f_2'}([\, \vec{f_1}(\vec{x})^T \;\; \vec{f_1'}(\vec{x})^T \,]^T)^T \,]^T$. The effect of the depth-growth here is adding to the output the result of a function on the first-hidden layer outputs.

3.2 Training of the Grown ANN

For an ANN grown from an original ANN with any combination of the methods described in the previous section, training is relatively straightforward, and can be done with any conventional iterative gradient-minimization method if continuous and differentiable nonlinearities are used in all the nodes. The gradient vector of the parameters can be computed directly on the grown ANN, since each partial derivative is evaluated independently. To keep the parameters of the original ANN frozen, they can simply be skipped during the parametric update in each iteration. Therefore, training can occur in-place with the desired training dataset without significant changes to the method. The growth method often re-

quires a smaller trainable parameter vector than constructing an ANN from scratch to achieve similar levels of empirical error. Thus training often progresses faster as well.

4 EXPERIMENTS AND RESULTS

In this chapter, the concepts and methods introduced in the previous chapters are put to test in two series of experiments, showcasing the capabilities and limitations of the ANN-growth method. Experiments were performed for both classification and regression tasks, and their results are discussed.

4.1 Classification

4.1.1 Data

For the classification experiments, two randomly generated datasets are used. Both datasets have two-dimensional input vectors, and binary target values in $\{-0.9, 0.9\}$. The usual target values of $\{-1, 1\}$ are not used since the ANNs to be trained on these datasets have the hypertangent function as their node-nonlinearities: because the hypertangent function approaches $\{-1, 1\}$ asymptotically as its input approaches $\{-\infty, \infty\}$, the weights of the output nodes could grow arbitrarily large if $\{-1, 1\}$ were used as the target values. By setting the magnitudes of these values to 0.9, the optimization procedure forces the weights to adopt reasonable values. The importance of this modification becomes apparent when the training of the growth nodes is considered later on.

The first dataset consists of 5000 samples, whose input vectors are uniformly drawn from $[-1, 1]^2$, and whose target value is defined as

$$y = \begin{cases} 0.9 & \text{if } \sqrt{x_0^2 + x_1^2} > 0.5, \\ -0.9 & \text{otherwise.} \end{cases} \tag{4.1}$$

Thus, the input space is divided into two classes with a circle of radius 0.5 centered at the origin as the class boundary. For reasons soon to be explicated, this dataset will hereafter be referred to as the reference dataset. See Fig. 4.1 for a visual representation.

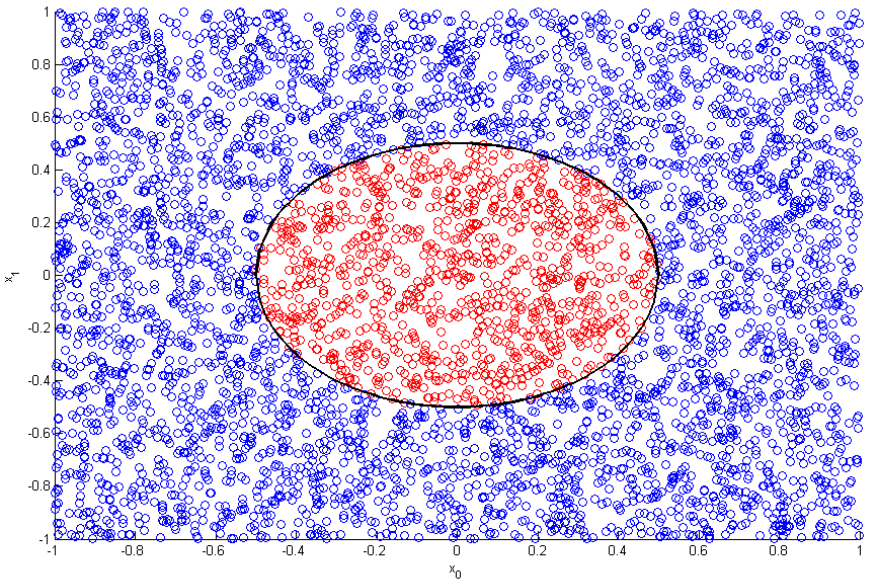

Figure 4.1: The reference dataset. Circles represent training samples. Black line shows class boundary.

The second dataset consists of only 100 samples uniformly drawn from the same input space as the reference dataset; however, the target values of the second dataset are defined by a slightly different function:

$$y = \begin{cases} 0.9 & \text{if } \sqrt{x_0^2 + x_1^2} > 0.5 \text{ or } x_1 < 0, \\ -0.9 & \text{otherwise.} \end{cases} \tag{4.2}$$

The class boundary separating the two classes in this case is the upper semicircle of the circular class boundary of the reference dataset. The second dataset will be called the test dataset from here on. See Fig. 4.2 for a visual representation.

4.1.2 Experiment

The goal of the classification experiment is to identify, as well as possible, the underlying class boundary upon which the test dataset is generated from the limited amount of test data available. To evince the performance of the ANN growth method, a conventional feed-forward, three-layer, 10-hidden-node, hypertangent ANN is first constructed and trained with the

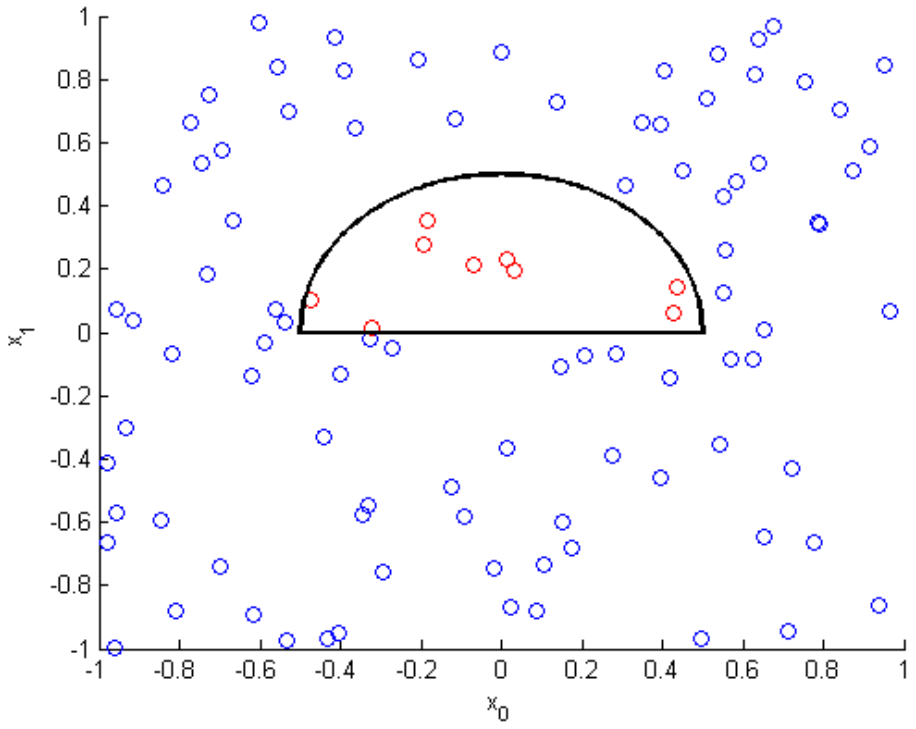

Figure 4.2: The test dataset. Circles represent training samples. Black line shows class boundary.

conjugate-gradient method to convergence on the test dataset, serving as a control group (henceforth referred to as the control ANN). Achieving a mean-squared error of 0.0049, the control ANN is able to completely separate the test dataset upon which it is trained. The classification boundary of the control ANN, however, deviates from the true underlying function by a good margin (Fig. 4.3).

Clearly, the control ANN is not able to take advantage of the fact that there is a very large reference dataset whose class boundary closely relates to that of the test dataset. However, without knowing the precise relationship, there is no way to incorporate that knowledge and use the large reference dataset directly in the construction and training of the control ANN. However, the application of ANN growth would be able to utilize the extra information embedded in the reference dataset.

First, the class boundary information of the reference dataset should be "encoded" in the form of an ANN. An ANN of the same structure as the control ANN is trained on the reference dataset. It is able to completely

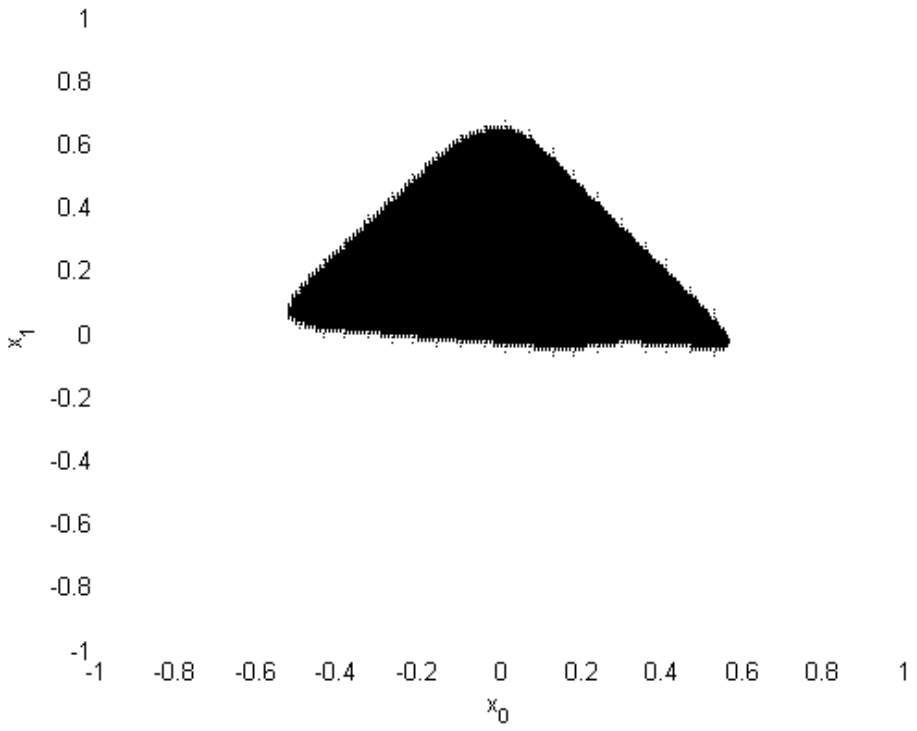

Figure 4.3: The classification boundary of the control ANN.

separate the reference dataset and yields the classification boundary seen in Fig. 4.4. It is henceforth referred to as the reference ANN.

Clearly, the classification boundary achieved by the reference ANN very closely approximates the true class boundary of the reference dataset, which is related to the class boundary of the test dataset; therefore, some information about the class boundary of the test dataset is now within the reference ANN. To use this information and adapt the reference ANN to classify the test dataset, one only needs to grow the ANN. The effects of different growths shall be explored in the subsequent paragraphs.

Breadth-growths

First experiments were conducted with only breadth-growths considered. First one, then two nodes were grown onto the first hidden layer of the reference ANN, and then trained to convergence on the test dataset. The ANN with only one breadth-growth yielded a mean-squared error of 0.1734, while the ANN with two breadth-growths yielded 0.0832. It can

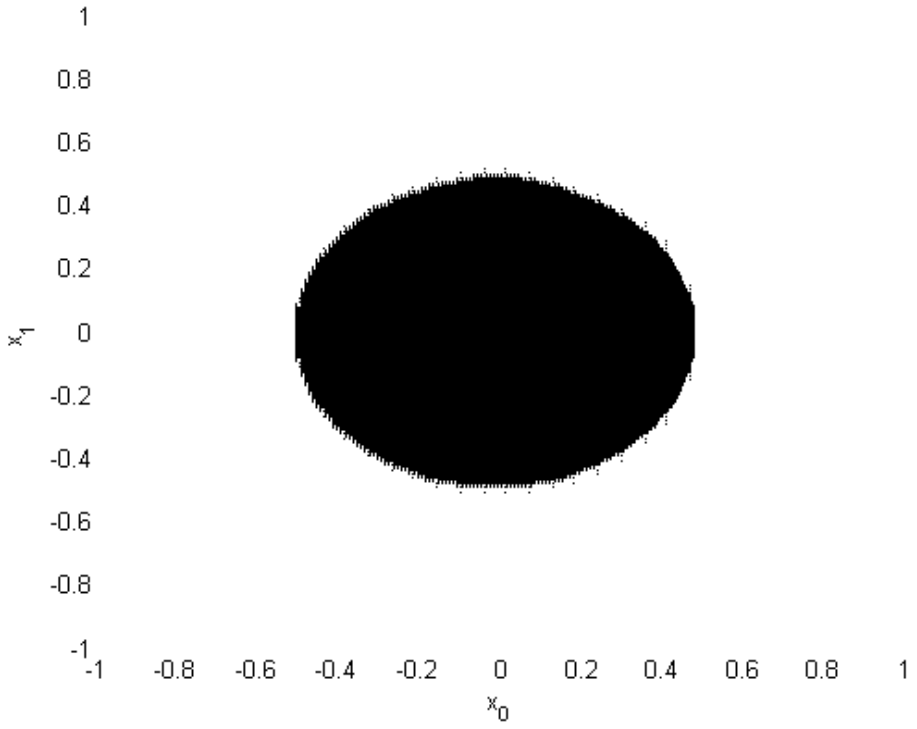

Figure 4.4: The classification boundary of the reference ANN.

be seen, however, that for both of these cases, the resultant classification boundaries were very different from the true class boundary; this is due to the lack of adaptation flexibility in using breadth-growths alone: they can divide the input space into finer regions (clearly demonstrated in the defined classification boundary lines in Fig. 4.5 and 4.6), but they lack the ability to modify the ANN output based on feature-mapping. Breadth-growths alone do not yield satisfactory results.

Depth-growths

Observing the behavior of the classification boundaries of the ANNs with only breadth-growths, it is clear that additional flexibility in adapting the reference ANN is required. Namely, the ability to remap the classification of specific Voronoi regions formed by the first hidden layer is required to change the assignment of the lower semicircle to 0.9 without affecting the classification of the rest of the input space. To achieve this goal, a depth-growth is performed on the reference ANN, with one additional node in

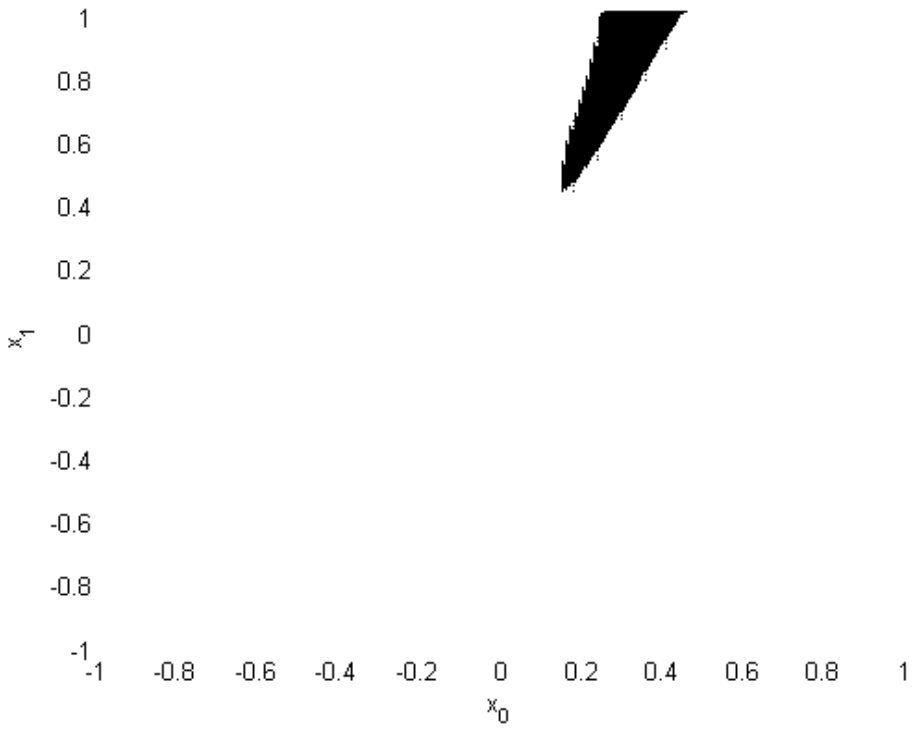

Figure 4.5: The classification boundary implemented by ANN with one breadth-growth.

the new hidden layer, in addition to the breadth-growths performed in the previous section. The resultant ANNs are then trained to convergence on the test dataset. Their performances are as expected: the ANN with one breadth-growth and one depth-growth, with mean-squared error of 0.0050, begins to show a classification boundary somewhat resembling the true class boundary, while the ANN with two breadth-growths and one depth-growth, with mean-squared error of 0.0049, shows a reasonably good approximation of the true class boundary (Fig. 4.7 and 4.8).

To quantitatively evaluate the generalizability of the ANNs, a regular sampling of the input space is performed to create a set of input vectors covering the entire input space. The input space is sampled in each dimension with a period of 0.01, creating a 40401-point grid of input vectors. For each ANN, the output at every point is evaluated, and compared to the target value yielded by the true class boundary of the test dataset. The control ANN missed 1052 out of 40401 points, while the grown ANN with two breadth-growths and one depth-growth missed 932 out of 40401 points.

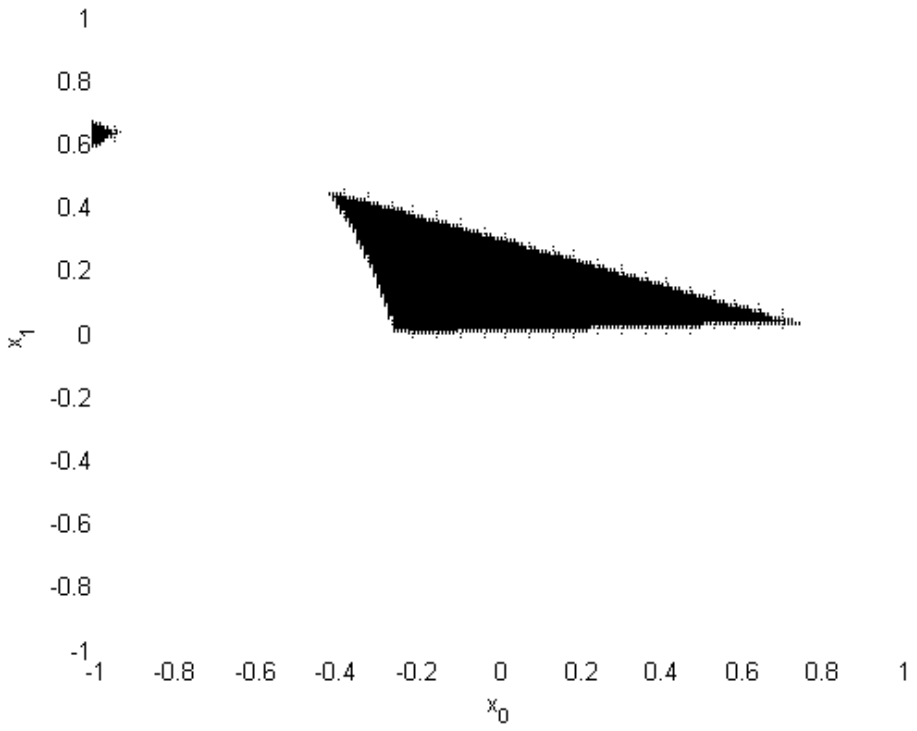

Figure 4.6: The classification boundary implemented by ANN with two breadth-growths.

This result confirms the prediction that since the grown ANN has fewer trainable parameters, it is likely to be more generalizable than an ANN trained from scratch at similar levels of empirical error.

4.1.3 Training considerations

When growing hypertangent ANNs like those used here, it is important to take care to avoid having the network parameter update caught in low-performance local minima. Since for grown ANNs much of the network would have already been trained to convergence, some weights could have very large values, and the gradient norm is usually small. If, during the training of the reference ANN, care is not taken to keep the parameters reasonably small, it is very likely that the grown nodes will be easily trapped in a remote local minimum, since the error propagating back from the output layer is affected by the gradient of the nonlinear function, which quickly approaches zero as the weight-values grow large. This is of less

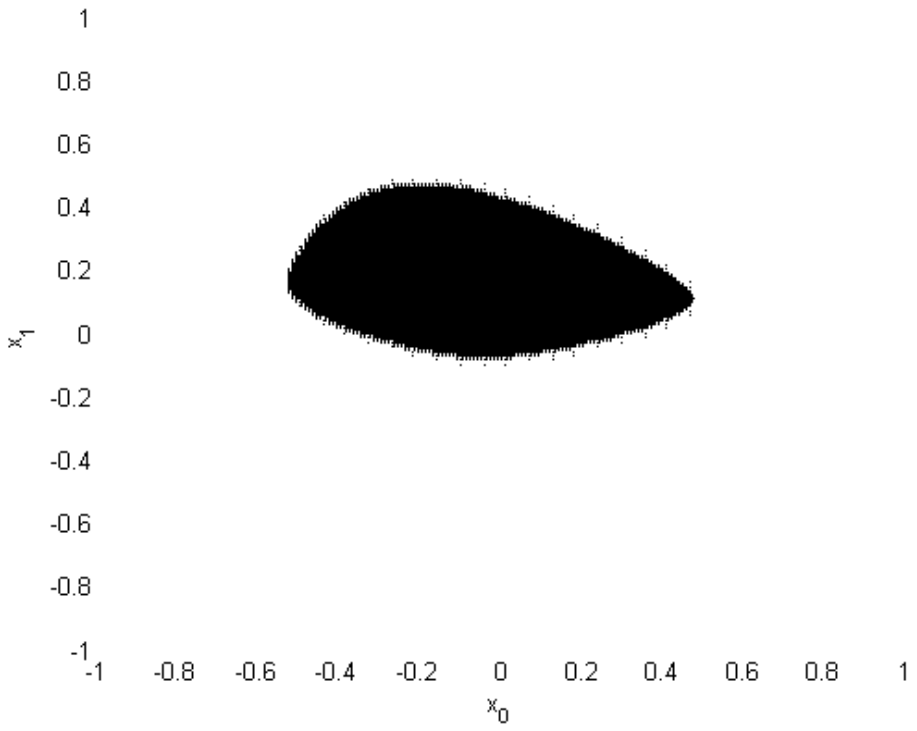

Figure 4.7: The classification boundary implemented by ANN with one breadth-growth and one depth-growth.

concern in regression tasks, where the output layer is linear; but for classification problems, it is advisable to employ the trick used in the previous sections, and set the target values to $\{-0.9, 0.9\}$ to prevent unchecked growth in the magnitudes of the weight values.

Due to the smaller number of trainable parameters, the grown ANN required fewer iterations to reach convergence. Training was stopped at 308 epochs, where the control ANN was trained to 1000 epochs. Their convergence paths are shown in Fig. 4.9.

4.2 Regression

4.2.1 Data

The regression experiment shares a similar setup with the classification setup. There are two distinct datasets, the reference dataset and the test dataset; however, whereas the classification experiment focuses on observ-

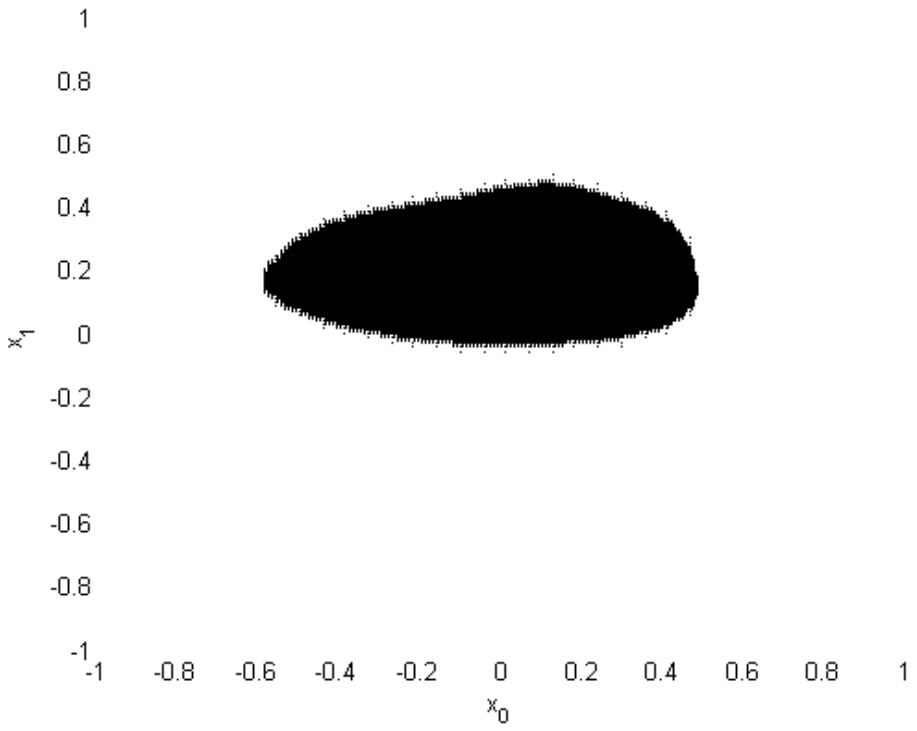

Figure 4.8: The classification boundary implemented by ANN with two breadth-growths and one depth-growth.

ing the performance differences between the control ANN and the grown ANN given limited data, this experiment focuses on the performance differences of these ANNs given noisy data. The reference dataset consists of 50 accurate samples, whose input vectors were randomly drawn from a uniform distribution on $[-1, 1]^2$. The target values of the reference dataset are computed according to the following function of the input vectors:

$$y = (x_0 + x_1)^2 \tag{4.3}$$

The test dataset, on the other hand, consists of 20 samples whose input vectors were also randomly drawn from a uniform distribution on $[-1, 1]^2$, but whose target values are computed from the function

$$y = \begin{cases} (x_0 + x_1)^2 + 1 & \text{if } x + y > 0, \\ (x_0 + x_1)^2 - 1 & \text{if } x + y < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

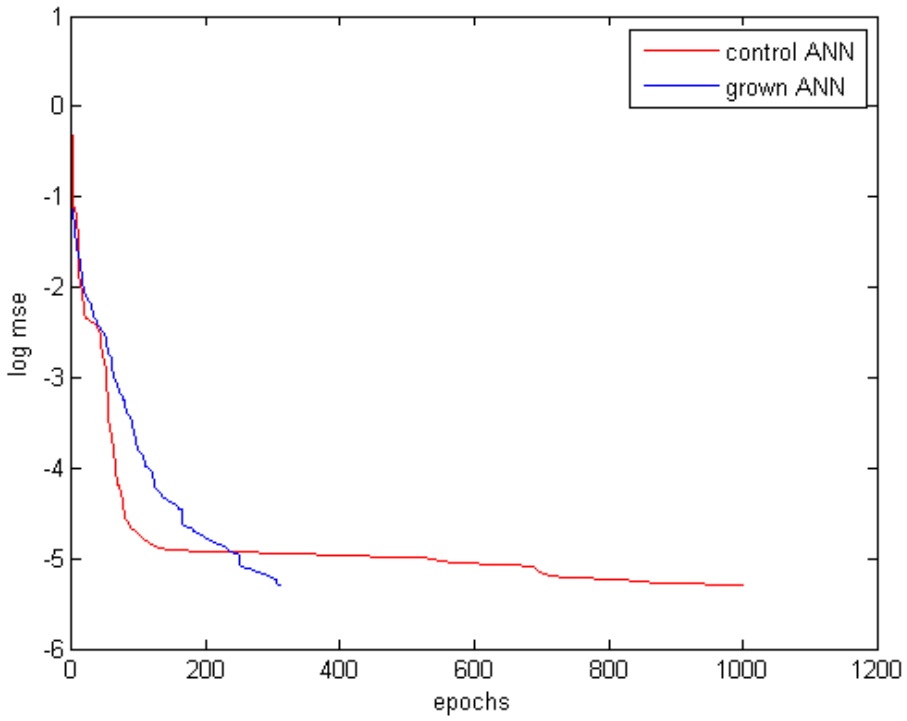

Figure 4.9: Training histories of the networks.

The target values are also injected with severe additive noise; a uniform noise in $[-0.25, 0.25]$ is added to the target values of all the samples in the test dataset.

4.2.2 Experiment

The experiment steps are similar to those of the classification experiment. First a feed-forward, three-layer, five-hidden-node, hypertangent ANN with linear output nodes (the control ANN) is trained with the conjugate-gradient algorithm to convergence on the test dataset. Then an ANN of the same structure (the reference ANN) is trained on the reference dataset to convergence. By observing the differences between the function of the test dataset and the function of the reference dataset, it is clear that the function of the test dataset is a modified version of the function of the reference dataset, with the modification being a function of some characteristic of the input vectors. Since no new characteristic of the input space needs to be defined, it is sufficient to perform depth-growths only on

the reference ANN. Two grown ANNs are trained, one with one additional node in the new hidden layer, and the other with two. As shown in Fig. 4.10, the ANN with one additional node does not have enough flexibility to modify the output to reflect the new functional relationship between the input vectors and the output. The ANN with two additional nodes,

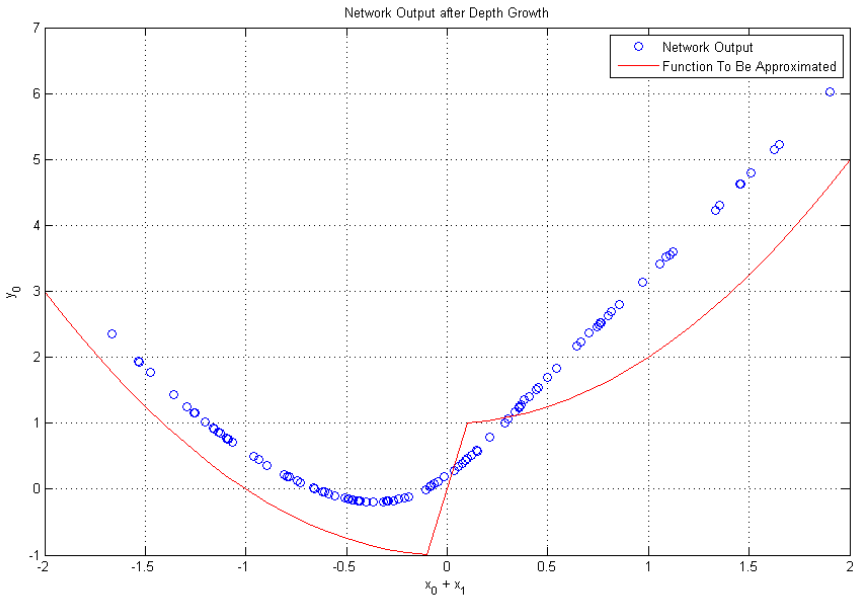

Figure 4.10: One-additional-node ANN outputs on random inputs.

on the other hand, performs very well, and is able to make the necessary adaptations to the function implemented by the reference network to closely approximate the functional relationship of the test dataset. Observe the overall performance comparison of the ANN with two depth-growths after training (Fig. 4.11). The test dataset (black triangles) is very noisy; the samples deviate greatly from their generating function (green line). The control ANN tries to follow the samples, so when the samples are far off from the function, the control ANN will be wrong. However, since the grown ANN only has two new nodes, its limited number of trainable parameters prevents the network from being able to follow the samples as well as the control ANN. The sum-squared error of the control ANN on the test dataset is 2.24, while the sum-squared error of the grown ANN is 3.38. But when a new test set is generated from Eq. (4.4) with 100 randomly drawn input vectors from a uniform distribution on $[-1, 1]^2$ and without

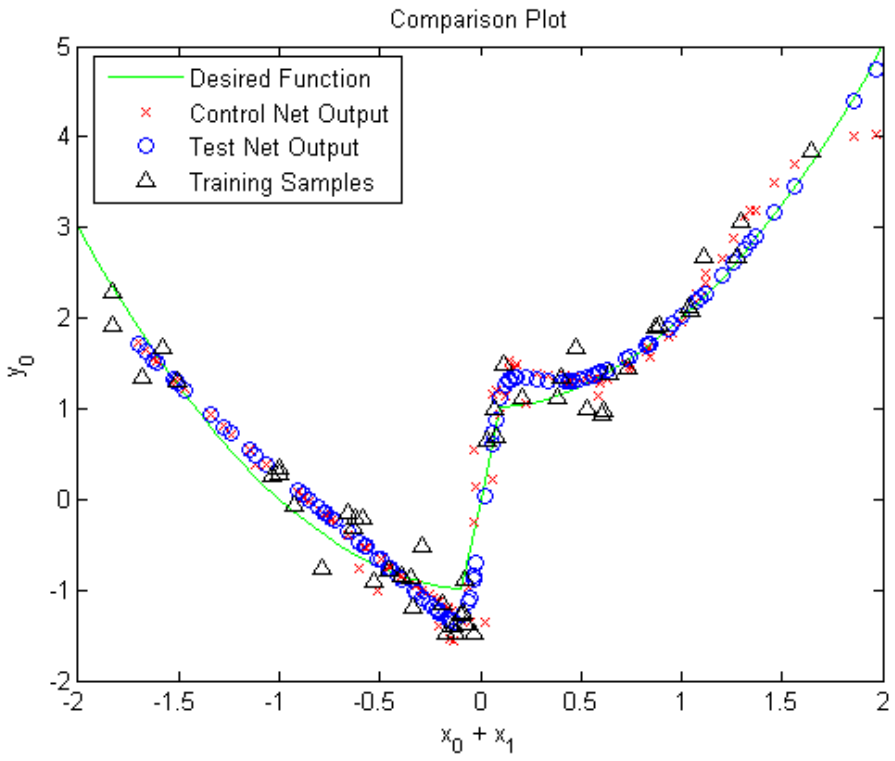

Figure 4.11: Two-additional-node ANN vs. control ANN.

any noise, the results are more than reversed: the sum-squared error of the control ANN on this new test set is 3.97, while that of the grown ANN is 2.08. As predicted, the reduction in noise-sensitivity of the grown ANN is apparent from this demonstration.

5 CONCLUSION

ANN training, like other forms of density estimation, is a constant struggle between presumption and humility. One does not wish to be overly presumptuous and state what he does not know, nor does he wish to be excessively humble and dismiss what truths he has come to learn. This dilemma lies at the heart of dependency estimation given empirical data. While sigmoidal ANNs are universal approximators capable of approximating any function arbitrarily well with sufficient number of hidden nodes, increasing the number of nodes without enough training samples cannot guarantee that the ANN learns the correct function. This struggle is manifested mathematically in the form of the bias/variance trade-off of estimators. The more accurately the estimator describes the training dataset, the more likely it is for the estimator to suffer high levels of performance degradation on data it has not seen. Without other means of assistance, given a limited training dataset, the accuracy on that dataset needs to be limited as well for the generalizability of the estimator to remain intact.

This thesis proposes a way to circumvent this trade-off by enlisting aid from similar problem domains. Analogous to the fact that if one has learned to write with a pen, he can also write with a pencil (though he might break several tips before he realizes that he cannot apply too much pressure, he will not become incapable of producing legible writing), by borrowing knowledge from an estimator built for another similar task and adapting it to fit the task at hand, the estimator can be trained more quickly, and with a lower data requirement. The paper demonstrated that for ANNs, this can be realized by growing nodes onto a trained ANN.

Growing nodes is attractive because one can obtain a large ANN on a small dataset without worrying about overfitting, since the bulk of the ANN would have been trained on another, larger dataset, and only a small number of trainable parameters are present from the growths. It is shown that this is theoretically desirable, since it is proven that the empirical

error becomes a better estimate of the total error for a smaller number of trainable parameters. This effect is observed in both the classification and the regression experiments: the grown ANNs are better able to estimate the sample-generating functions, and thus are more generalizable, than the control ANNs.

For niche tasks and problems where no extensive data is available, with the growth method in mind, one can look for related, more general problems where more data collection has been done. If the problem domains are similar enough, large ANNs trained on those problems can be grown to perform the task at hand. If the number of parameters in the necessary growths is smaller than the number of parameters in an ANN created from scratch, it is always more desirable to train and use the grown ANN, since it is guaranteed to be more generalizable if similar levels of empirical performance with ANNs trained from scratch are obtained.

REFERENCES

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[2] V. N. Vapnik, *Estimation Dependences Based on Empirical Data*. New York, NY: Springer-Verlag, 1982.

[3] A. R. Barron, "Statistical properties of artificial neural networks," in *IEEE International Conference on Decision and Control*, 1989, pp. 280–285.

[4] A. R. Barron, "Approximation and estimation bounds for artificial neural networks," *Machine Learning*, vol. 14, pp. 115–133, 1994.

[5] L. Devroye, "Automatic pattern recognition: A study of the probability of error," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 530–543, 1988.

[6] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[7] A. R. Barron and T. M. Cover, "Minimum complexity density estimation," *IEEE Transactions on Information Theory*, vol. 37, pp. 1034–1054, 1991.

[8] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and Vapnik-Chervonenkis dimension," *Journal of the Association for Computing Machinery*, vol. 36, no. 4, pp. 929–965, 1989.

[10] D. Haussler, "Generalizing the PAC model: Sample size bounds from metric dimension-based uniform convergence results," in *Symposium on Foundations of Computer Science*, 1989, pp. 40–45.

[11] N. Linial, Y. Mansour, and R. L. Rivest, "Results on learnability and the Vapnik-Chervonenkis dimension," in *Workshop on Computational Learning Theory*, 1988, pp. 56–68.

[12] D. Haussler, "Decision theoretic generalizations of the PAC model for neural net and other learning applications," *Information and Computation*, vol. 100, pp. 78–150, 1992.

[13] M. Talagrand, "Sharper bounds for Gaussian and empirical processes," *The Annuals of Probability*, vol. 22, no. 1, pp. 28–76, 1994.

[14] N. Sauer, "On the density of families of sets," *Journal of Combinatorial Theory, Series A*, vol. 13, pp. 145–147, 1972.

[15] P. Grünwald, "A tutorial introduction to the minimum description length principle," in *Advances in Minimum Description Length: Theory and Applications*, P. Grünwald, I. J. Myung, and M. A. Pitt, Eds. Cambridge, MA: MIT Press, April 2004.

[16] G. H. L. Cheang and A. R. Barron, "Estimation with two hidden layer neural nets," *Neural Networks*, vol. 1, pp. 375–378, 1999.

[17] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York, NY: John Wiley & Sons, Inc., 2001.

[18] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, vol. 1, 1993, pp. 293–299.

[19] A. Pelagotti and V. Piuri, "Entropic optimum synthesis of multilayered feed-forward anns," in *IEEE International Conference on Neural Networks*, vol. 1, 1995, pp. 253–258.

[20] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Los Altos, CA: Morgan Kaufmann, 1990, pp. 524–532.

[21] X. Wang, D. Brown, T. Haynes, and T. M. J. Hui, "A new method in incremental neural network construction by using boosting," in *IEEE International Symposium on Intelligent Signal Processing*, 2003, pp. 173–177.

[22] K. M. R. Alam and M. M. Islam, "Combining boosting with negative correlation learning for training neural network ensembles," in *International Conference on Information and Communication Technology*, 2007, pp. 68–71.

[23] L. Fletcher, V. Katkovnik, F. E. Steffens, and A. P. Engelbrecht, "Optimizing the number of hidden nodes of a feedforward artificial-neural network," in *IEEE International Joint Conference on Neural Networks*, vol. 2, 1998, pp. 1608–1612.