MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY
and
CENTER FOR BIOLOGICAL AND COMPUTATIONAL LEARNING
DEPARTMENT OF BRAIN AND COGNITIVE SCIENCES

A.I. Memo No. 1438 May, 1996
C.B.C.L. Paper No. 116

A Formulation for Active Learning with Applications to Object Detection

Kah Kay Sung and Partha Niyogi

This publication can be retrieved by anonymous ftp to publications.ai.mit.edu.

Abstract

We discuss a formulation for active example selection for function learning problems. This formulation is obtained by adapting Fedorov's optimal experiment design to the learning problem. We specifically show how to *analytically* derive example selection algorithms for certain well defined function classes. We then explore the behavior and sample complexity of such active learning algorithms. Finally, we view object detection as a special case of function learning and show how our formulation reduces to a useful heuristic to choose examples to reduce the generalization error.

Copyright © Massachusetts Institute of Technology, 1995

This report describes research done at the Center for Biological and Computational Learning and the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Center is provided in part by a grant from the National Science Foundation under contract ASC–9217041.

1 Introduction

Many problems in disparate fields like computer vision, finance, natural language processing are increasingly being approached from a machine learning perspective. Typically, there is some task to be performed like object recognition/detection or stock market prediction and the learner has access to data relevant to this task. On the basis of this data set the learner develops hypotheses and uses these to perform the task.

In most classical formulations of learning from examples, the data (examples) are assumed to be randomly drawn and presented to the learner. This is the case for for a variety of situations ranging from network models [16, 14], PAC [23] frameworks, and classical pattern recognition. In this sense, the learner is a *passive* recipient of information about the target concept.

In contrast, one could consider a learner that plays a more *active* role in collecting its examples. In this paper, we take a formal look at the problem of selecting high utility examples for machine learning systems. The example selection problem falls under a newly emerging general area of research, called *active learning*, that investigates how learners can pose intelligent queries to teachers under various learning scenarios, to achieve "better" learning results. Active learning differs from traditional example-based learning paradigms in the following way: Rather than passively accepting training examples that randomly describe a target concept, an *active* learner uses information derived from its current state and prior knowledge about the target concept to intelligently gather useful examples from specific input space locations for further training. By carefully generating intelligent queries instead of performing random sampling, one can expect active learning techniques to have faster learning rates and better approximation results than traditional example-based learning algorithms.

Our main focus is on active example selection strategies for a *function approximation* based learning framework. Specifically, we address the following three questions:

1. Given a function approximation based learning task and some prior information about the target function, are there principled strategies for selecting useful training data in some "optimal" fashion?

2. Assuming such principled data selection strategies do exist, do these active strategies require fewer examples than classical learning techniques to approximate target functions to the same degree of accuracy?

3. Can one directly apply these active example selection strategies to real-world function approximation learning tasks or easily adapt them into more feasible forms without losing too much of their original flavor?

Using ideas from Optimal Experiment Design [6], we begin by proposing an active example selection formulation for function approximation tasks and show that one can indeed select high utility examples for a given task in a principled and "optimal" fashion. MacKay [10] and Cohn [5] have adopted a similar formalism for active learning and we comment on differences with their work at appropriate points in the future. More recently, Sollich [19] has arrived at a very similar formulation motivated by statistical mechanics.

While the formulation proposed (and the variants suggested by others) are certainly well motivated, they are however, computationally intractable in general. In this paper, we show that the general formulation can be used to analytically derive precise, tractable, data selection algorithms for three specific function approximation classes: (1) unit step functions, (2) polynomial approximators and (3) Gaussian radial basis function networks. In addition, we describe some conditions on parameterized function classes for which the general formulation can be analytically solved to yield active algorithms. For the three function classes considered in this paper, we provide either theoretical or empirical results suggesting that the active strategy learns the target function with fewer data examples than random sampling.

Ultimately, the litmus test of any theoretical framework is whether it can affect the way practical learning systems are built. To this effect, we consider a reduced version of the original active learning formulation that essentially hunts for new data where approximation "error bars" are high. We show how such a scheme, with minor modifications, leads to a practical example selection strategy (referred to henceforth as a "boot-strap" strategy). We have adopted this method of choosing examples in an object and pattern class detection approach. Although the "boot-strap" strategy loses some of the original active learning flavor and may thus be "sub-optimal" in its choice of new examples, we show empirically that it still outperforms random sampling in training a frontal face detection system, and is therefore still an effective means of dealing with unmanageably large data sets to make learning tasks tractable.

2 Background and Approach

We start with an overview of *active learning* and related work. Active learning has appeared in various forms throughout *knowledge engineering* and *machine learning* literature. One early implementation can be found in certain expert systems, where an important component of learning relies on issuing queries to the instructor. For example, Sammut and Banerji [17] use queries about specific examples as part of a strategy for efficiently learning a target concept. Shapiro's Algorithmic Debugging System prompts the user with a variety of query types to locate errors in Prolog programs [18]. In computational learning theory, several types of *active learning* queries have also been defined (see for example [3]) and compared with Valiant's *probably approximately correct* (PAC) model of concept identification under random sampling [24]. Angluin [2], for example, has shown that there are concept classes that can be efficiently learnt with membership and equivalence queries, but not with random sampling in Valiant's PAC model.

Some early connectionist approaches toward active learning include: Ahmad and Omohundro [1] on training networks by selective attention; Hwang et. al. [9] on a query-based neural network learning scheme that generates queries near classification boundaries; Plutowski and White [13] on an efficient feedforward network training technique that selects new training examples with maximum potential utility from among available candidate examples. Both Hwang et. al. [9] and Plutowski et. al. [13] choose new training examples according to information derived from a partially trained network.

Plutowski and White [13] examines the learning task from a more general function approximation standpoint, viz., approximating a target function, $g(\vec{x})$, using a network output function, $F(\vec{w}, \vec{x})$, parameterized by weights $\vec{w}$. They design their criteria for selecting new examples to meet two objectives: (1) to maximize the accuracy of fit between network output, $F(\vec{w}, \vec{x})$, and the target function, $g(\vec{x})$, and (2) to minimize the approximation's unreliability in the presence of noise. The paper quantifies the above two considerations by proposing an *Integrated Mean Squared Error* (IMSE) measure to be minimized:

$$\begin{aligned} IMSE(\mathbf{x}^n) &= \int\int [g(\vec{x}) - F(l_n(\mathbf{x}^n, \mathbf{y}^n), \vec{x})]^2 \gamma^n(d\mathbf{y}^n|\mathbf{x}^n)\mu(d\vec{x}) \\ &= \int E[(g(\vec{x}) - F(\vec{w_n}, \vec{x}))^2|\mathbf{x}^n]\mu(d\vec{x}), \end{aligned}$$

where $\mathbf{x}^n, \mathbf{y}^n$ are the current n pairs of input-output training examples, $l_n(\mathbf{x}^n, \mathbf{y}^n) = \vec{w}_n$ is the learning rule that begets network weights $\vec{w}_n$ from the training examples, $\gamma^n(\mathbf{y}^n|\mathbf{x}^n)$ is the conditional probability of output distribution $\mathbf{y}^n$ given input distribution $\mathbf{x}^n$, $E[\cdot|\mathbf{x}^n]$ is conditional expected network output mean squared error given input distribution $\mathbf{x}^n$, and μ is the probability distribution over the input space $\vec{x}$. To select the next training example, the learning algorithm samples at the next input location $\vec{x_{n+1}}$ that maximally decreases the $IMSE$. Unfortunately, an obvious problem with the approach is that both the $IMSE$ and the analytic expression for its decrement (not shown) assume a *known* target function $g(\vec{x})$. This is seldom a reasonable assumption in real learning scenarios where the target function is *unknown*.

2.1 Regularization Theory and Function Approximation — A Review

Our main focus in this chapter is on function approximation based active learning. We briefly review *regularization theory* as a lead in to our active learning formulation.

Let $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points obtained by sampling a function g, possibly in the presence of noise, where d is the input dimensionality. The function approximation task is to recover g, or at least obtain a reasonable estimate of it, by means of an approximator $\hat{g}$. Clearly, the problem is ill-posed [8] because there can be an infinite number of functions that pass through those data points. Some constraints are thus needed to transform the problem into a well-posed one. The *regularization* approach [21] [22] [12] [4] selects a function $\hat{g}$ that minimizes the following functional:

$$H[\hat{g}] = \sum_{i=1}^{n} (y_i - \hat{g}(\vec{x_i}))^2 + \lambda \parallel P\hat{g} \parallel^2 . \tag{1}$$

The first term of Equation 1 penalizes discrepancies between the solution, $\hat{g}$, and the observed data. The second term, usually called a *stabilizer*, embodies a priori knowledge about the *smoothness* of the solution. P is a constraint operator, usually a linear differential operator, and $\| \cdot \|$ stands for a *norm* on the function space containing $\hat{g}$, usually the L_2 norm. Together, they favor functions that do not vary too quickly on $\Re^d$. The *regularization parameter*, λ, determines the trade-off between the two terms — data reliability and prior beliefs. Poggio and Girosi have shown that the solution to Equation 1 has the following simple form:

$$\hat{g}(\vec{x}) = \sum_{i=1}^{n} c_i G(\vec{x}; \vec{x_i}) + p(\vec{x}), \tag{2}$$

where G, p and the coefficients c_i, can all be derived from the constraint operator P, the n data points $(\vec{x_i}, y_i)$, the stabilizer and some boundary conditions (see [14] for details).

For our purpose, it is convenient to adopt a probabilistic interpretation of *regularization* that treats the function $\hat{g}$ and the data set $\mathcal{D}_n$ as random, dependent variables (see [15]). Using Bayes rule, we can express the conditional probability of the function $\hat{g}$ given examples $\mathcal{D}_n$, $\mathcal{P}(\hat{g}|\mathcal{D}_n)$, in terms of the prior probability of $\hat{g}$, $\mathcal{P}(\hat{g})$, and the conditional probability of $\mathcal{D}_n$ given $\hat{g}$, $\mathcal{P}(\mathcal{D}_n|\hat{g})$:

$$\mathcal{P}(\hat{g}|\mathcal{D}_n) \propto \mathcal{P}(\mathcal{D}_n|\hat{g})\mathcal{P}(\hat{g}). \tag{3}$$

Equation 3 relates to the regularization functional of Equation 1 as follows: Suppose noise at each of the n data points is *identically independently Gaussian distributed* with variance σ^2. The conditional probability, $\mathcal{P}(\mathcal{D}_n|\hat{g})$, can be written as:

$$\mathcal{P}(\mathcal{D}_n|\hat{g}) \propto \exp\left(- \sum_{i=1}^{n} \frac{1}{2\sigma^2} (y_i - \hat{g}(\vec{x_i}))^2 \right).$$

Similarly, if $\hat{g}$ is a stochastic process [11] [7], we can write $\mathcal{P}(\hat{g})$ as:

$$\mathcal{P}(\hat{g}) \propto \exp\left(-l \parallel P\hat{g} \parallel^2\right),$$

where l is some fixed constant, P and $\parallel \cdot \parallel$ are as defined earlier. Equation 3 thus becomes:

$$\begin{aligned}\mathcal{P}(\hat{g}|\mathcal{D}_n) &= Ke^{-\sum_{i=1}^{n}\frac{1}{2\sigma^2}(y_i-\hat{g}(\vec{x_i}))^2}\exp\left(-l \parallel P\hat{g} \parallel^2\right) \\ &= K\exp\left(-[\sum_{i=1}^{n}\frac{1}{2\sigma^2}(y_i-\hat{g}(\vec{x_i}))^2 + l \parallel P\hat{g} \parallel^2]\right)\end{aligned}$$

where K is some fixed constant. Taking natural logarithms on both sides and performing some additional algebra yields:

$$-2\sigma^2 \ln \mathcal{P}(\hat{g}|\mathcal{D}_n) + \ln K = \sum_{i=1}^{n}(y_i - \hat{g}(\vec{x_i}))^2 + 2\sigma^2 l \parallel P\hat{g} \parallel^2,$$

which is identically Equation 1 with $\lambda = 2\sigma^2 l$ and $H[\hat{g}] = -2\sigma^2 \ln \mathcal{P}(\hat{g}|\mathcal{D}_n) + \ln K$. So, by choosing a function $\hat{g}$ that minimizes $H[\hat{g}]$, *regularization* essentially maximizes the conditional probability $\mathcal{P}(\hat{g}|\mathcal{D}_n)$. In other words, it chooses:

$$\begin{aligned}\hat{g} \in \arg\min_f H[f] &= \arg\max_f \mathcal{P}(f|\mathcal{D}_n) \\ &= \arg\max_f \mathcal{P}(\mathcal{D}_n|f)\mathcal{P}(f),\end{aligned}$$

that is, an a-posteriori most probable function $\hat{g}$ given the set of examples $\mathcal{D}_n$.

2.2 A Bayesian Framework

The *active learning* problem for function approximation can be posed as follows: Let $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points sampled from an unknown target function g, possibly in the presence of noise, where d is the input dimensionality. Given an approximation function concept class, $\mathcal{F}$, where each $f \in \mathcal{F}$ has prior probability $\mathcal{P}_{\mathcal{F}}(f)$, one can use *regularization* techniques to approximate g from $\mathcal{D}_n$ (in the Bayes optimal sense) by means of a function $\hat{g} \in \mathcal{F}$. We want a strategy to determine at what input location one should sample the next data point, $(\vec{x_{n+1}}, y_{n+1})$, in order to obtain the "best" possible Bayes optimal approximation of the unknown target function g with our concept class $\mathcal{F}$.

One can use ideas from *optimal experiment design* [6] to approach the active data sampling problem in two stages:

1. **Define what we mean by the "best" possible Bayes optimal approximation of an unknown target function.** We propose an optimality criterion for evaluating the "goodness" of a solution with respect to an *unknown* target function, similar in spirit to the cost function, Equation 1, for a *known* target.

2. **Formalize mathematically the task of determining where in input space to sample the next data point.** We express the above mentioned optimality criterion as a cost function to be minimized, and the task of choosing the next sample as one of minimizing the cost function with respect to the input space location of the next sample point.

Earlier work by Cohn [5] and MacKay [10] have tried using similar *optimal experiment design* techniques to collect data with maximum information about the target function. Our work here differs from theirs in two respects. First, we use a different, and perhaps more general, optimality criterion for evaluating solutions to an unknown target function. Specifically, our optimality criterion considers both bias and variance components in the solution's *output* generalization error. In contrast, both MacKay and Cohn use a "less complete" optimality criterion that favors solutions with only small variance components in *model parameter* space. Second, we also examine the important sample complexity issue, i.e., does the active strategy require fewer examples than random sampling to approximate the target to the same degree of accuracy? After completion of this work, we learnt that Sollich [19] had also recently developed a similar formulation to ours, but his analysis is conducted in a statistical physics framework. We will review these differences in greater detail in a later section.

3 The Active Learning Formulation

In order to optimally select examples for a learning task, one should first have a clear notion of what an "ideal" learning goal is for the task. One can then measure an example's utility in terms of how well the example helps the learner achieve the goal, and devise an active sampling strategy that selects examples with maximum potential utility. In this section, we propose one such learning goal — to find an approximation function $\hat{g} \in \mathcal{F}$ that "best" estimates the *unknown* target function g. We then derive an example utility cost function for the goal and finally present a general procedure for selecting examples.

3.1 An Optimality Criterion for Learning an Unknown Target Function

Let g be the target function that we want to estimate by means of an approximation function $\hat{g} \in \mathcal{F}$. If the target function g were known, then one natural measure of how well (or badly) $\hat{g}$ approximates g would be their *Integrated Squared Difference* (ISD) over the input space, $\Re^d$, or over some appropriate region of interest:

$$\delta(\hat{g}, g) = \int_{\vec{x} \in \Re^d} (g(\vec{x}) - \hat{g}(\vec{x}))^2 d\vec{x}. \tag{4}$$

In most function approximation tasks, the target g is unknown, so we clearly cannot express the quality of a learning result in terms of g. We propose an alternative scheme for characterizing probabilistically the quality of an approximation result that takes into account only $\hat{g}$, the approximation function itself, and the example data points it approximates, without actually having to know g. Here, our objective notion is similar in spirit to the *integrated squared difference* "misfit" criterion described above. We elaborate further on what we mean below:

Figure 1(a) shows two approximation functions, $\hat{g}_1$ and $\hat{g}_2$, for a set of data points, $\mathcal{D}$, from an *unknown* target function g. Without further knowledge of the target function, g, one would normally guess that $\hat{g}_1$ is a more probable (and hence better) hypothesis for g, because it oscillates less between the data points. This aspect of an approximation function's "goodness" has been fully captured by *regularization*, which assigns $\mathcal{P}(\hat{g}_1|\mathcal{D})$ a higher likelihood value than $\mathcal{P}(\hat{g}_2|\mathcal{D})$.

Figure 1(b) shows a function, $\hat{g}$, that approximates two unknown target functions g_1 and g_2, sampled at $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively. Notice that in this example, the approximator $\hat{g}$ fits both

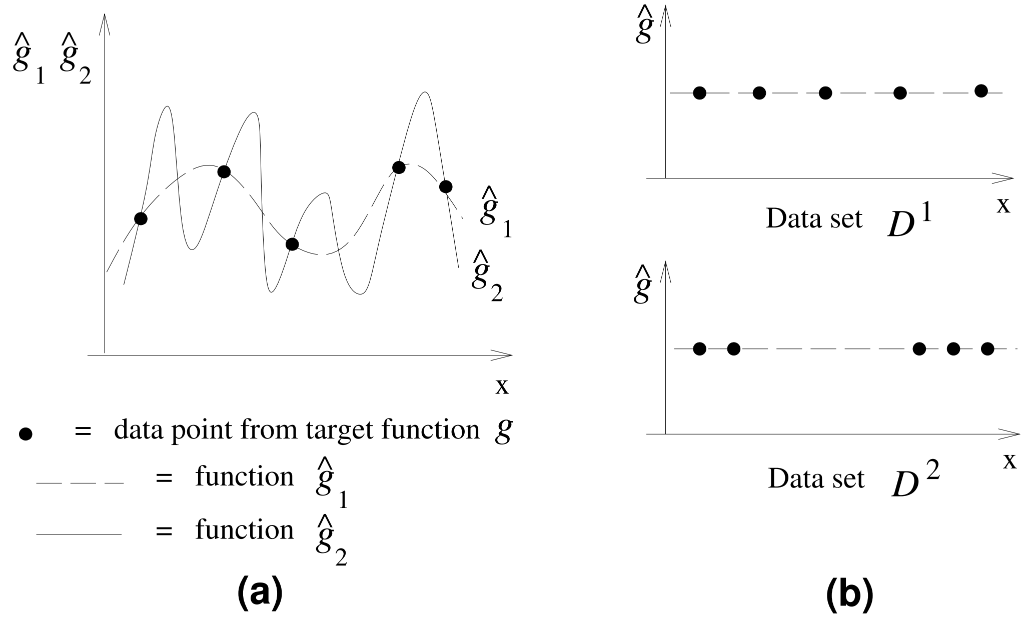


Figure 1: **(a):** Two approximation functions, $\hat{g_1}$ and $\hat{g_2}$, for the same set of data points sampled from an unknown target function. $\hat{g_1}$ oscillates less between the data points, so one would normally guess that it is a more probable hypothesis for the unknown target function. **(b):** An approximation function, $\hat{g}$, for two sets of data points sampled from two (possibly different) unknown target functions. Although $\hat{g}$ fits both sets of data points exactly, one might still expect it to more closely resemble the unknown target function of the top system than of the bottom system. This is due to the uneven example distribution in the bottom system.

data sets exactly, so we have $\hat{g} = \arg\max_{f \in \mathcal{F}} \mathcal{P}(f|\mathcal{D}^1)$ and $\hat{g} = \arg\max_{f \in \mathcal{F}} \mathcal{P}(f|\mathcal{D}^2)$. Intuitively however, one might still expect the actual *misfit* between g_1 and $\hat{g}$ to be smaller than the actual *misfit* between g_2 and $\hat{g}$. This is because $\mathcal{D}^1$ is a more representative data sample for g_1 than $\mathcal{D}^2$ is for g_2, and in both systems, $\hat{g}$ is directly derived from $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively. One can view this *expected misfit* notion between an *unknown* target g and its approximation function $\hat{g}$, as a sense of "uncertainty" that one has in the current solution. The notion is not captured by the *regularization* framework, and as we shall see, depends instead on the distribution of training examples over the input space.

Since our active learning task is to determine the best input space location for sampling next, a reasonable learning goal would be to sample at locations that minimize the *expected misfit* notion between the unknown target g and the resulting approximation $\hat{g}$.

3.2 Evaluating a Solution to an Unknown Target — The Expected Integrated Squared Difference

We now formalize the above *expected misfit* notion as a mathematical functional to be minimized. The general idea is as follows: Let $\mathcal{F}$ be the approximation function class in our learning task. Suppose we treat the unknown target function g as a random variable in $\mathcal{F}$, then one way of determining the *expected misfit* between the *regularized* solution, $\hat{g}$, and the unknown target function, g, would be to compute an expected version of some difference measure between them, such as their *integrated squared difference*, $\delta(\hat{g}, g)$ (see Equation 4). Taking into account $\mathcal{D}_n$, the n data points seen so far, and $\mathcal{P}_{\mathcal{F}}(g)$, the prior probability of g in $\mathcal{F}$, we have the following a-posteriori likelihood for g: $\mathcal{P}(g|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)$. The *expected* integrated squared difference (EISD) between an unknown target, g, and its estimate, $\hat{g}$, given $\mathcal{D}_n$, is thus:

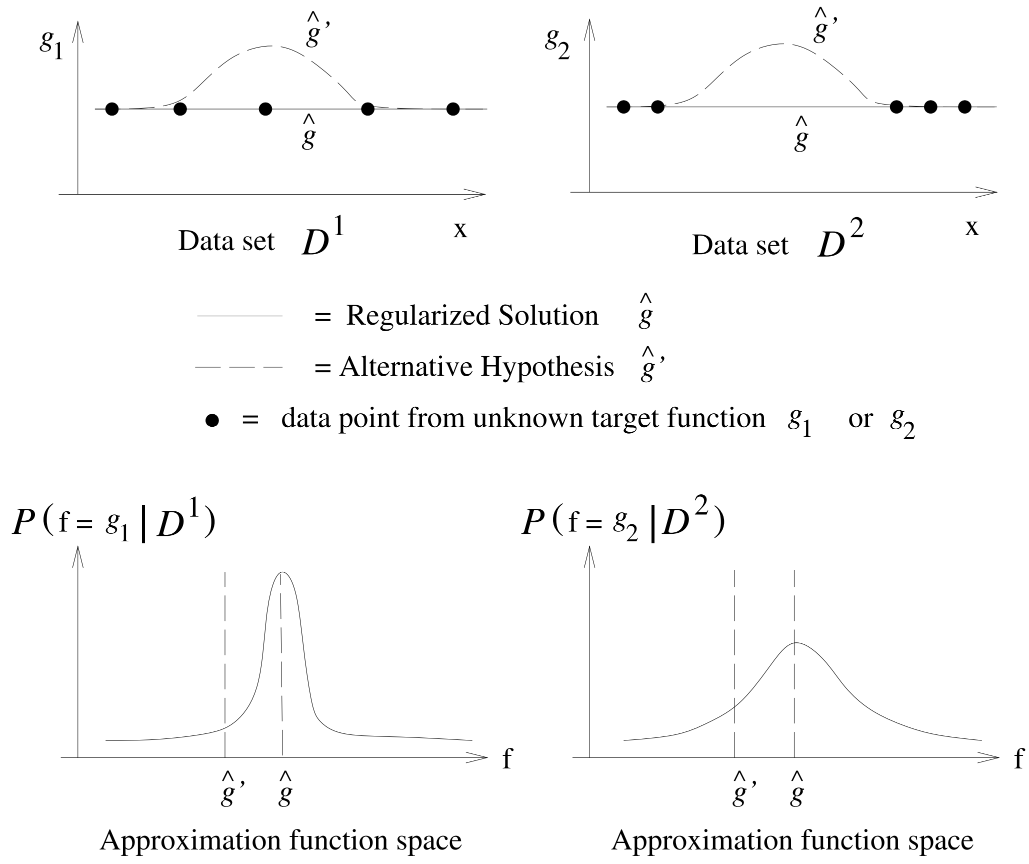


Figure 2: **Top Row:** A regularized solution, $\hat{g}$, for two unknown target functions, g_1 (left graph) and g_2 (right graph). The curve $\hat{g}'$ is an alternative hypothesis for the two unknown target functions. There is more evidence against $\hat{g}'$ being a true hypothesis for g_1 than for g_2 because the first system has a data point near the center of the input space where $\hat{g}'$ differs considerably from $\hat{g}$ and the data point. **Bottom Row:** Graphs depicting the a-posteriori probability distribution of the *unknown* target in approximation function space for the two systems. Because there is more evidence against alternative hypotheses like $\hat{g}'$ in the first system than in the second system, we get a sharper peak for the a-posteriori distribution at $\hat{g}$ in the first system than in the second system.

$$E_{\mathcal{F}}[\delta(\hat{g}, g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g}, g)dg = \int_{g\in\mathcal{F}} \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)\delta(\hat{g}, g)dg. \tag{5}$$

The EISD is intuitively pleasing as an "uncertainty" measure for evaluating a solution to an *unknown* target, because its value decreases with better distributed data samples. The following example illustrates how the measure agrees well with "human intuition". We return to the two function approximation problems described in Figure 1(b). In the first system, one intuitively expects a smaller discrepancy between the unknown target and its approximation function than in the second system, even though the same regularized estimate $\hat{g}$ fits both data sets equally well. This is because the data samples $\mathcal{D}^1$ in the first system are more evenly (and hence better) distributed than the samples $\mathcal{D}^2$ in the second system. We now argue that the EISD measure in the first system should indeed be smaller than the EISD measure in the second system.

Consider the same two systems in the top row of Figure 2, where $\hat{g}$ is the regularized solution for the two unknown target functions g_1 and g_2. The two unknown targets are sampled at $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively, and both data sets contain the same number of data points. Consider next an alternate hypothesis $\hat{g'}$ for g_1 and g_2, that differs slightly from the regularized solution $\hat{g}$ over some region of the input space. Because the first system has better distributed data points than the second system, there is more evidence *against* most alternative hypotheses like $\hat{g'}$ being a viable solution for g_1 than for g_2. Mathematically, this means that for most alternative hypotheses like $\hat{g'}$, the ratio $\mathcal{P}(g_1 = \hat{g'}|\mathcal{D}^1)/\mathcal{P}(g_1 = \hat{g}|\mathcal{D}^1)$ is smaller than the ratio $\mathcal{P}(g_2 = \hat{g'}|\mathcal{D}^2)/\mathcal{P}(g_2 = \hat{g}|\mathcal{D}^2)$. One can therefore expect $E_{\mathcal{F}}[\delta(\hat{g}, g_1)|\mathcal{D}^1] < E_{\mathcal{F}}[\delta(\hat{g}, g_2)|\mathcal{D}^2]$, which agrees well with "human intuition". The bottom row of Figure 2 depicts the difference between the two systems graphically. Because most alternative hypotheses are poor solutions for the first data set $\mathcal{D}^1$, the first unknown target g_1 has an a-posteriori probability distribution that is heavily weighted around $\hat{g}$ in approximation function space. The same is less true about the a-posteriori probability distribution for g_2 in the second system. Thus, $\hat{g}$ is a more "stable", and hence a more "certain" solution for g_1 than for g_2.

3.3 Selecting the Next Sample Location

Let g be the unknown target function that we want to learn, $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be the set of n examples seen so far, and $\hat{g_n}$ be the current regularized approximation for g. We now formalize the task of determining the best input space location to sample next. Since our learning goal is to minimize the *expected misfit* between g and its regularized solution, a reasonable sampling strategy would be to choose the next example from the input location $\vec{x_{n+1}} \in \Re^d$ that minimizes the EISD between g and its new estimate $\hat{g_{n+1}}$.

How does one predict the new EISD that results from sampling the next data point at location $\vec{x_{n+1}}$? Suppose we also know the target output value (possibly noisy), y_{n+1}, at $\vec{x_{n+1}}$. The EISD between g and its new estimate $\hat{g}_{n+1}$ would then be $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$, where $\hat{g_{n+1}}$ can be recovered from $\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})$ via regularization. In reality, we do not know y_{n+1}, but we can derive its conditional probability distribution from $\mathcal{D}_n$, the data samples seen so far. Once again, let $\mathcal{F}$ be the approximation function class for our learning task and $\mathcal{P}_{\mathcal{F}}(f)$ be the prior probability of f in $\mathcal{F}$, then:

$$\mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) \propto \int_{f \in \mathcal{F}} \mathcal{P}(\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})|f)\mathcal{P}_{\mathcal{F}}(f)df. \tag{6}$$

Because y_{n+1} is a random variable and not a fixed value as we had assumed earlier, this leads to the following *expected* value for the new EISD, if we sample our next data point at $\vec{x_{n+1}}$:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{-\infty}^{\infty} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]dy_{n+1}. \tag{7}$$

Notice from Equation 5 that $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$ in the above expression is actually independent of the unknown target function g, and so $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ (henceforth referred to as the *total output uncertainty*) is fully computable from available information in the learning model. Clearly, the optimal input location to sample next is the location that minimizes $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$, i.e.:

$$\hat{\vec{x_{n+1}}} = \arg\min_{\vec{x_{n+1}}} \mathcal{U}(g_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}). \tag{8}$$

3.4 Summary of the Active Learning Procedure

We summarize the key steps involved in our active learning strategy for finding the optimal next sample location:

1. Compute $\mathcal{P}(g|\mathcal{D}_n)$. This is the a-posteriori likelihood of the different functions g given $\mathcal{D}_n$, the n data points seen so far.

2. Assume a new point $\vec{x_{n+1}}$ to sample.

3. Assume a value y_{n+1} for this $\vec{x_{n+1}}$. One can compute $\mathcal{P}(g|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1}))$ and hence the *expected* integrated squared difference (EISD) between the target and its new estimate $\hat{g_{n+1}}$. This is given by $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$ (see Equation 5).

4. At the assumed $\vec{x_{n+1}}$, y_{n+1} has a probability distribution given by Equation 6. Averaging the resulting EISD over all y_{n+1}'s, we obtain the *total output uncertainty* for $\vec{x_{n+1}}$, given by $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ in Equation 7.

5. Sample at the input location $\hat{\vec{x_{n+1}}}$ that minimizes the *total output uncertainty* cost function $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$.

Some final remarks about our example selection strategy: Intuitively, a reasonable selection criterion should choose new examples that provide dense information about the target function g. Furthermore, the choice should also take into account the learner's current state, namely $\mathcal{D}_n$ and $\hat{g_n}$, so as to maximize the *net* amount of information gained. Our scheme treats an approximation function's *expected misfit* with respect to the unknown target g (i.e. their EISD) as a measure of *uncertainty* in the current solution. It selects new examples, based on the data that it has already seen, to minimize the *expected* value of the resulting EISD measure. In doing so, it essentially maximizes the net amount of information gained with each new example.

Our main results in this active learning formulation are:

1. a cost function that captures the *expected misfit* optimality criterion (Equation 5) for evaluating the solution to an *unknown* target function, and

2. a formal specification for the task of selecting new training examples with maximum potential utility (Equation 8).

The developed framework, and the associated observations may, in themselves, be interesting from a theoretical standpoint, but in practice, another fundamental concern must also be addressed — the computational complexity issue. Both Equations 5 and 8, though theoretically computable from available information in the learning model, are clearly intractable in their current form. Nevertheless, we maintain the formulation still serves as a possible "optimal" benchmark for evaluating other active example selection schemes. Later in this paper, we shall consider a reduced version of the original function approximation based active learning formulation that essentially hunts for new data where approximation "error bars" are high. We also show how such a scheme, with minor modifications, leads to a "boot-strap" example selection strategy we have adopted to useful advantage in an object and pattern class detection approach that we have developed.

3.5 Previous Frameworks Revisited

At this point, we are in a position to make concrete the differences between the technique developed here and those adopted by MacKay [10] and Sollich [19]. Recall that our active formulation requires the computation of two expectations. One is over the a-posteriori distribution over the function space $\mathcal{F}$, the other is over the a-posteriori distribution over the space of y_{n+1}'s one would expect given the data and a proposed sample location (x_{n+1}). Specifically,

$$\mathcal{U}(x_{n+1}) = E_{P(y_{n+1}|D_n,x_{n+1})} E_{P(g|D_n \cup (x_{n+1},y_{n+1}))} \delta(g, \hat{g})$$

Here g ranges over all the functions in $\mathcal{F}$ and $\hat{g}$ is the MAP solution that the learner would use in practice in such a Bayesian setting.

In contrast, MacKay uses the following criterion:

$$\mathcal{U} = E_{P(y|D_n,x_{n+1})} E_{P(\vec{a}|D_n \cup (x_{n+1},y_{n+1}))} \parallel \vec{a} - \vec{a}_\mu \parallel$$

He assumes that $\mathcal{F}$ is a parameterized family with parameters denoted by $\vec{a}$. Correspondingly, $\vec{a}_\mu$ is the mean parameter value (averaged over the a-posteriori density over parameter space, i.e., $\vec{a}_\mu = E_{P(\vec{a}|D_n \cup (x_{n+1},y_{n+1}))} \vec{a}))$). Notice that his criterion operates in the parameter space rather than the true function space. If, closeness in parameter space is linearly proportional to closeness in function space, then the criterion would be equivalent to the following (where $\bar{g}$ is the mean over the function space):

$$E_{P(y_{n+1}|D_n,x_{n+1})} E_{P(g|D_n \cup (x_{n+1},y_{n+1}))} \delta(g, \bar{g})$$

Finally, Sollich uses the following criterion:

$$\mathcal{U} = E_{P(y_{n+1}|D_n,x_{n+1})} E_{P(g|D_n \cup (x_{n+1},y_{n+1}))} E_{P(h|D_n \cup (x_{n+1},y_{n+1}))} \delta(g, h)$$

He distinguishes the space of target functions $\mathcal{F}$ from the space of hypothesis functions (say $\mathcal{H}$). One could then compute the a-posteriori distributions over both the target space and the hypothesis space given a set of data. These a-posteriori distributions are used to compute three averages in the sense above.

Having now reproduced each of the above frameworks in a common, consistent notation, we make the following observations.

First, it is clear that the three frameworks differ slightly. MacKay's criterion, which computes "spread" around the mean of the a-posteriori distribution in parameter space is the variance of the a-posteriori distribution in a strictly correct sense. He ignores the bias. Sollich's criterion (assuming $\mathcal{F} = \mathcal{H}$) reduces to ours if one assumes that $P(h|D)$ is a Kronecker-delta centered on $\hat{g}$, the MAP solution. The two inner expections reduce essentially to just one non-trivial one; this condition is referred to as "deterministic" learning as opposed to the stochastic "Gibbs" learning framework seemingly consistent with the statistical mechanics approach. We have preferred to use the MAP solution in our framework since in practice, that is what is used in most learning systems.

Finally, like Sollich, we use the output generalization error (δ) in our approach. In reality, one is interested in choosing examples to reduce the total generalization error. Using a metric based on the parameter space is only an indirect measure of this generalization error. In the case of function classes sufficiently non-linear in their parameters this is likely to give suboptimal performance.

We now return to the main subject of investigation, i.e., sample complexity, analytical tractability, and practical utility.

4 Comparing Sample Complexity

To demonstrate the usefulness of the above active learning procedure, we show analytically and empirically that the active strategy learns target functions with fewer data examples than random sampling for three specific approximation function classes: (1) unit step functions, (2) polynomial approximators and (3) Gaussian radial basis function networks. For all three function classes, one can derive exact analytic data selection algorithms following the key steps outlined in Section 3.4.

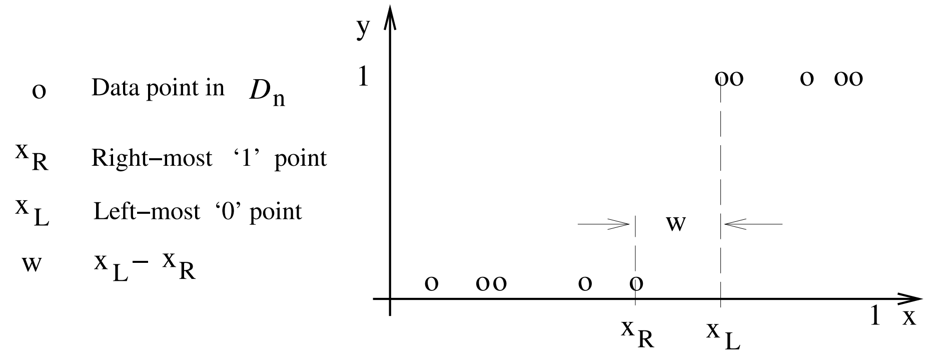


Figure 3: Diagram showing the notation used for our unit-step example.

4.1 Unit Step Functions

We first consider the following simple class of one-dimensional unit-step functions described by a single parameter a which takes values in $[0, 1]$. Let us denote the unit-step function by:

$$u(x-a) = \begin{cases} 1 & \text{if } x \geq a \\ 0 & \text{otherwise} \end{cases}$$

The target and approximation function class for this problem is given by:

$$\mathcal{F} = \{u(x-a) | 0 \leq a \leq 1\}$$

Assuming a has an a-priori uniform distribution on $[0, 1]$, we obtain the following prior distribution on the approximation function class:

$$\mathcal{P}_{\mathcal{F}}(g = u(x-a)) = \begin{cases} 1 & \text{if } 0 \leq a \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Suppose we have a *noiseless* data set, $\mathcal{D}_n = \{(x_i, y_i); i = 1, ..n\}$, consistent with some unknown target function $g = u(x-a)$ that the learner has to approximate. We want to find the best input location to sample next, $x \in [0, 1]$, that would provide us with maximal information. Let x_R be the right most point in $\mathcal{D}_n$ whose y value is 0, i.e., $x_R = \max_{i=1,..n}\{x_i | y_i = 0\}$ (see Figure 3). Similarly, let $x_L = \min_{i=1,..n}\{x_i | y_i = 1\}$ and $w = x_L - x_R$. Following the general procedure outlined in Section 3.4, we go through the following steps:

1. Derive $\mathcal{P}(g|\mathcal{D}_n)$. One can show that:

$$\mathcal{P}(g = u(x-a)|\mathcal{D}_n) = \begin{cases} \frac{1}{w} & \text{if } a \in [x_R, x_L] \\ 0 & \text{otherwise} \end{cases}$$

2. Suppose we sample next at a particular $x \in [0, 1]$, we would obtain y with the distribution:

$$P(y = 0|\mathcal{D}_n, x) = \begin{cases} \frac{(x_L - x)}{x_L - x_R} = \frac{(x_L - x)}{w} & \text{if } x \in [x_R, x_L] \\ 1 & \text{if } x \leq x_R \\ 0 & \text{otherwise} \end{cases}$$

$$P(y=1|\mathcal{D}_n,x) = \begin{cases} \frac{(x-x_R)}{x_L-x_R} = \frac{(x-x_R)}{w} & \text{if } x \in [x_R, x_L] \\ 1 & \text{if } x \geq x_L \\ 0 & \text{otherwise} \end{cases}$$

3. For a particular y, the new data set would be $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (x,y)$ and the corresponding EISD can be easily obtained using the distribution $\mathcal{P}(g|\mathcal{D}_{n+1})$. Averaging this over $\mathcal{P}(y|\mathcal{D}_n,x)$ as in **step 4** of the general procedure, we obtain:

$$\mathcal{U}(g_{\hat{n+1}}|\mathcal{D}_n,x) = \begin{cases} \frac{w^2}{12} & \text{if } x \leq x_R \text{ or } x \geq x_L \\ \frac{1}{12w}((x_L-x)^3 + (x-x_R)^3) & \text{otherwise} \end{cases}$$

4. Clearly the new input location that minimizes the *total output uncertainty*, $\mathcal{U}(g_{\hat{n+1}}|\mathcal{D}_n,x)$, measure is the midpoint between x_L and x_R:

$$x_{\hat{n+1}} = \arg\min_{x\in[0,1]} \mathcal{U}(g_{\hat{n+1}}|\mathcal{D}_n,x) = \frac{x_L+x_R}{2}.$$

Thus, by applying the general procedure to this trivial case of one-dimensional unit-step functions, we get the familiar binary search learning algorithm that queries the midpoint of x_R and x_L. For this function class, one can show analytically in PAC-style [23] that this active data sampling strategy takes fewer examples to learn an unknown target function to a given level of *total output uncertainty* than randomly drawing examples according to a uniform distribution in x.

Theorem 1 *Suppose we want to collect examples so that we are guaranteed with high probability (i.e. probability $> 1-\delta$) that the* total output uncertainty *is less than ϵ. Then a passive learner would require at least $\frac{1}{\sqrt{48\epsilon}}\ln(1/\delta)$ examples while the active strategy described earlier would require at most $(1/2)\ln(1/12\epsilon)$ examples.*

4.2 Polynomial Approximators

We consider next a univariate polynomial target and approximation function class with maximum degree K, i.e.:

$$\mathcal{F} = \{g(x,\vec{a}) = g(x,a_0,\ldots,a_K) = \sum_{i=0}^{K} a_i x^i\}.$$

The model parameters to be learnt are $\vec{a} = [a_0\, a_1\, \ldots\, a_K]^{\mathrm{T}}$ and x is the input variable. We obtain a prior distribution for $\mathcal{F}$ by assuming a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$:

$$\mathcal{P}_{\mathcal{F}}(g(\cdot,\vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}). \tag{9}$$

Our task is to approximate an *unknown* target function $g \in \mathcal{F}$ within the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ on the basis of sampled data. Let $\mathcal{D}_n = \{(x_i, y_i = g(x_i) + \eta)|i = 1, \ldots, n\}$ be a *noisy* data sample from the unknown target in the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$, where η is an additive zero-mean Gaussian noise term with variance σ_s^2. We compare two different ways of selecting the next data point: (1) sampling the function at a random point x according to a uniform distribution in $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ (i.e. passive learning), and (2) using our active learning framework to derive an exact algorithm for determining the next sampled point.

4.2.1 The Active Strategy

Here, we go through the general active learning procedure outlined in Section 3.4 to derive an exact expression for $\hat{x_{n+1}}$, the next query point. We summarize the key derivation steps below:

1. Let $\bar{\mathbf{x}}_\mathbf{i} = [1 \; x_i \; x_i^2 \; \ldots \; x_i^K]^\mathtt{T}$ be a power vector of the $i^{\mathbf{th}}$ data sample's input value. One can show (see Appendix A.1.1) that the a-posteriori approximation function class distribution, $\mathcal{P}(\vec{a}|\mathcal{D}_n)$, is a multivariate Gaussian centered at $\hat{\vec{a}}$ with covariance Σ_n, where:

$$\hat{\vec{a}} = \Sigma_n \left(\frac{1}{\sigma_s^2} \sum_{i=1}^{n} \bar{\mathbf{x}}_\mathbf{i} y_i \right)$$

   and:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2} \sum_{i=1}^{n} \left(\bar{\mathbf{x}}_\mathbf{i} \bar{\mathbf{x}}_\mathbf{i}^T \right). \tag{10}$$

2. Deriving the *total output uncertainty* expression $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x_{n+1})$ requires several steps (see Appendix A.1.2 and A.1.3). Taking advantage of the Gaussian distribution on both the parameters $\vec{a}$ and the noise term, we eventually get:

$$\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x_{n+1}) = |\Sigma_{n+1}\mathbf{A}| \propto |\Sigma_{n+1}|, \tag{11}$$

   where $\mathbf{A}$ is a constant $(K+1) \times (K+1)$ matrix of numbers whose $(i,j)^{\mathbf{th}}$ element is:

$$\mathbf{A}_{i,j} = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} t^{(i+j-2)} dt$$

   Σ_{n+1} has the same form as Σ_n and depends on the previous data, the priors, noise and the next sample location x_{n+1}. When minimized over x_{n+1}, we get $\hat{x_{n+1}}$ as the maximum utility location where the active learner should next sample the unknown target function.

4.2.2 Simulations — Error Rate versus Number of Examples

We perform some simulations to compare the active strategy's sample complexity with that of a passive learner which receives uniformly distributed random training examples on the input domain $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$. In this experiment, we investigate whether our active example selection strategy learns an unknown target to a smaller average error rate than the passive strategy for the same number of data samples. The experiment proceeds as follows:

We randomly generate 1000 target polynomial functions using a fixed Gaussian prior on the model parameters $\vec{a} = [a_0\, a_1\, \ldots\, a_K]^{\mathrm{T}}$. For each target polynomial, we collect data sets with noisy y values ranging from 3 to 50 samples in size, using both the active and passive sampling strategies. We then assume the same Gaussian priors on the approximation function class to obtain a regularized estimate of the target polynomial for each data set. Because we know the actual target polynomial for each data set, one can compute the actual integrated squared difference between the target and its estimate as an approximation error measure. We compare the two sampling strategies by separately averaging their approximation error rates for each data sample size over the 1000 different target polynomials.

In our simulations, we use polynomials of maximum degree $K = 9$, distributed according to the following independent Gaussian priors on model parameters: for each a_j in $\vec{a} = [a_0\, a_1\, \ldots\, a_9]^{\mathrm{T}}$, we have:

$$\mathcal{P}(a_j) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp\left(-\frac{a_j^2}{2\sigma_j^2}\right),$$

where $\sigma_j = 0.9^{j+1}$. In other words, $\Sigma_{\mathcal{F}}$ of Equation 9 is a 10×10 diagonal covariance matrix such that:

$$\Sigma_{\mathcal{F}}(i,j) = \begin{cases} \sigma_{i-1}^2 = 0.9^{2i} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Qualitatively, our priors favor smooth functions by assigning higher probabilities to polynomials with smaller coefficients, especially for higher powers of x. We also fix the input domain to be $[x_{\mathtt{LO}}, x_{HI}] = [-5, 5]$.

Figure 4 shows the average *integrated squared difference* between the 1000 randomly generated target polynomials and their regularized estimates for different data sample sizes. We repeated the same simulations three times, each with a different output noise variance in the data samples: $\sigma_s = 0.1$, 1.0 and 5.0. Notice that the active strategy has a lower average error rate than the passive strategy particularly for smaller data samples. From this experiment, one can conclude empirically that our active sampling strategy learns with fewer data samples than random sampling even when dealing with noisy data.

4.2.3 Simulations — Incorrect Priors

So far we have assumed that the approximation function class $\mathcal{F}$ that the Bayesian learner uses is identical to the target class: in other words, the learner has correct prior knowledge about the target. This is highly questionable in a real world situation where little is known about the target function and its properties and the learner is most likely to have an inaccurate prior model. We now investigate how the active learning strategy behaves if the this is the case. Specifically, we consider the following three scenarios:

In the first case, the active learner assumes a higher polynomial degree with similar but slightly larger Gaussian variances than the true target priors. We use a 9^{th} degree (i.e. $K = 9$) polynomial function class with Gaussian priors $\sigma_j = 0.9^{j+1}$ to approximate an unknown 8^{th} degree (i.e. $K = 8$) target polynomial with Gaussian priors $\sigma_j = 0.8^{j+1}$. Qualitatively, the approximation function class is *more complex* and favors smooth estimates *less strongly* than the target class.

The second case deals with the exact opposite scenario. The active learner uses a *lower* degree polynomial with similar but slightly smaller Gaussian variances ($K = 7$ and $\sigma_j = 0.7^{j+1}$) to

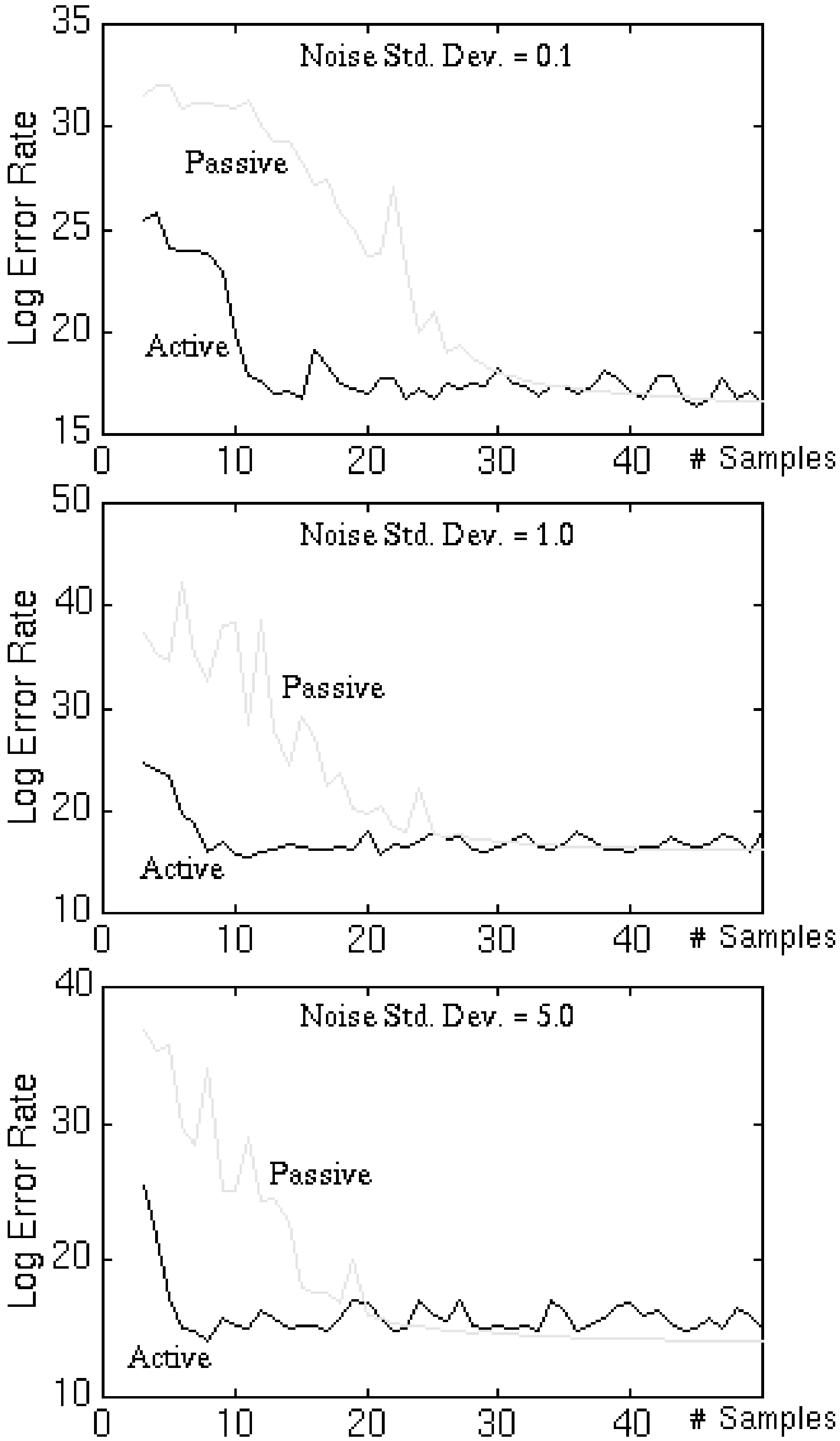


Figure 4: Comparing active and passive learning average error rates at different output noise levels for polynomials of maximum degree $K = 9$. We use the same priors on the target and approximation function classes. The three graphs above plot log error rates against number of samples. See text for detailed explanation. The dark and light curves are the active and passive learning error rates respectively.

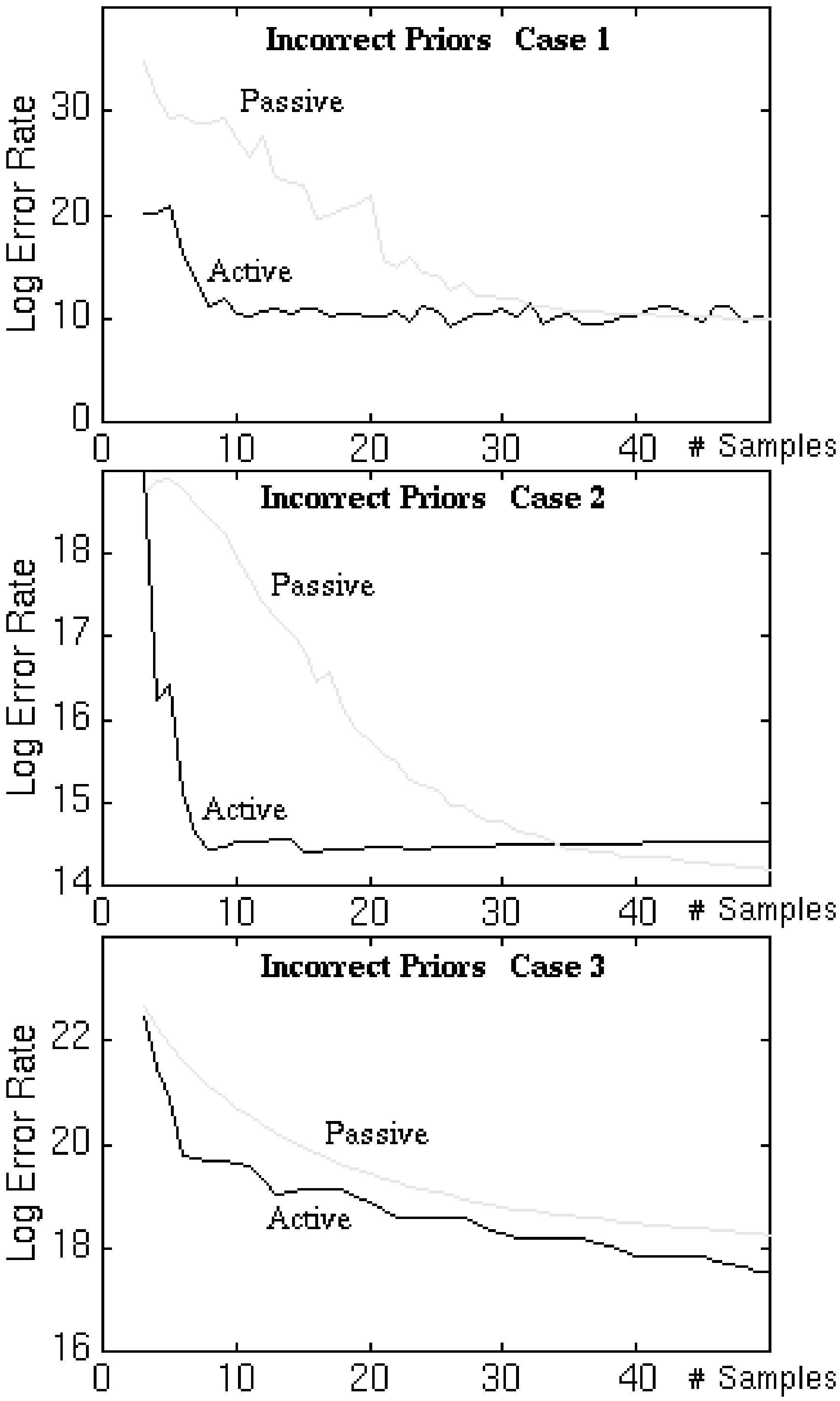


Figure 5: Comparing active and passive learning average error rates for slightly different priors between the target and approximation function classes. **Top:** Results for the first case. The approximation function class uses a *higher* degree polynomial with *larger* Gaussian variances on its coefficients ($K = 9$ and $\sigma_j = 0.9^{j+1}$) versus ($K = 8$ and $\sigma_j = 0.8^{j+1}$). **Middle:** The approximation function class uses a *lower* degree polynomial with smaller Gaussian variances on its coefficients ($K = 7$ and $\sigma_j = 0.7^{j+1}$) versus ($K = 8$ and $\sigma_j = 0.8^{j+1}$). **Bottom:** The approximation and target polynomial function classes have smoothness priors that differ in form. In all three cases, the active learning strategy still results in lower approximation error rates than random sampling for the same number of data points.

approximate an unknown 8^{th} degree (i.e. $K = 8$) target with Gaussian priors $\sigma_j = 0.8^{j+1}$. Here, the approximation function class is *less complex* and favors smooth estimates *more strongly* than the target class.

In the third case, we consider a polynomial approximation function class $\mathcal{F}$ whose prior distribution has a different form from that of the target class. Let $p \in \mathcal{F}$ be a polynomial in the approximation function class. One can quantify the overall "smoothness" of p by integrating its squared first derivative over the input domain $[x_{\text{LO}}, x_{\text{HI}}]$:

$$\mathcal{Q}(p(\cdot, \vec{a})) = \int_{x_{\text{LO}}}^{x_{\text{HI}}} \left[\frac{dp(x, \vec{a})}{dx}\right]^2 dx. \tag{13}$$

The "smoothness" measure above leads to a convenient prior distribution on $\mathcal{F}$ that favors smoothly varying functions:

$$\mathcal{P}_{\mathcal{F}}(p) \propto \exp(-\mathcal{Q}(p(\cdot, \vec{a}))) \exp(-\frac{a_0^2}{2\sigma_0^2}).$$

Here, a_0 is the constant term in the polynomial p, whose coefficients are $\vec{a} = [a_0\, a_1\, \ldots\, a_K]^{\mathrm{T}}$. Although a_0 does not affect the "smoothness" measure in Equation 13, we impose on it a Gaussian distribution with variance σ_0^2 so $\mathcal{P}_{\mathcal{F}}(p)$ integrates to 1 over all polynomials in $\mathcal{F}$ like a true probability density function. One can show (see Appendix A.1.4 for detailed derivation) that $\mathcal{P}_{\mathcal{F}}(p)$ has the following general form similar to Equation 9, the priors on polynomials with independent Gaussian distributed coefficients:

$$\mathcal{P}_{\mathcal{F}}(p(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}).$$

The new covariance term $\Sigma_{\mathcal{F}}$ is as given below:

$$\Sigma_{\mathcal{F}}^{-1}(i,j) = \begin{cases} 1/\sigma_0^2 & \text{if } i = j = 1 \\ 2\frac{(i-1)(j-1)}{i+j-3}(x_{\text{HI}}^{i+j-3} - x_{\text{LO}}^{i+j-3}) & \text{if } 2 \leq i \leq K+1 \text{ and } 2 \leq j \leq K+1 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

For this third case, we use an 8^{th} degree (i.e. $K = 8$) polynomial function class with $\sigma_0 = 0.8$ in its "smoothness" prior to approximate a target polynomial class of similar degree with Gaussian priors $\sigma_j = 0.8^{j+1}$. Although the approximation and target function classes have prior distributions that differ somewhat in form, both priors are qualitatively similar in that they favor smoothly varying polynomials.

For all three cases of slightly incorrect priors described above, we compare our active learner's sample complexity with that of a passive learner which receives random samples according to a uniform distribution on $[x_{\text{LO}}, x_{\text{HI}}]$. We repeat the active versus passive function learning simulations performed earlier by generating 1000 target polynomials, collecting noisy data samples ($\sigma_s = 0.5$), computing regularized estimates, averaging and comparing their approximation errors in the same way as before. Figure 5 plots the resulting average *integrated squared difference* error rates over a range of data sample sizes for all three cases. Despite the incorrect priors, we see that the active learner still outperforms the passive strategy.

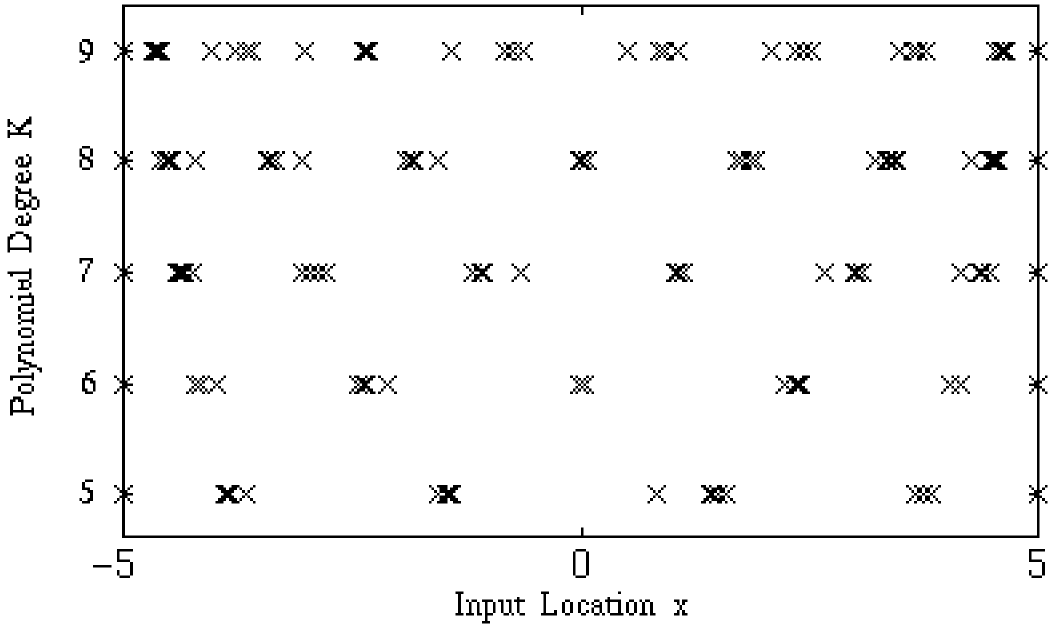


Figure 6: Distribution of the first 50 data points the active learner selects as the polynomial degree K of the approximation function class varies from 5 to 9. In all 5 cases, we assume a fixed data output noise level of $\sigma_s = 0.5$. Notice that for polynomials of degree K, the active learner clusters its data samples typically around $K + 1$ locations.

4.2.4 Distribution of Data Points

Notice from Equations 10, 11, 12 and 14 that the *total output uncertainty* measure $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ for polynomial approximators (i.e., Equation 11) does not depend on the previous y data values actually observed, but only on the previous input locations sampled. In other words, the previously observed y data values do not affect $\hat{x_{n+1}}$, the optimal location to sample next. One can show that this behavior is common to all approximation function classes that are linear in their model parameters [10] [19].

Given a polynomial approximation function class of maximum degree K, one can thus precompute the sequence of input locations that our active learner will sample to gain maximum information about the unknown target. There are two sampling trends that are noteworthy here. First, the active strategy does not simply sample the input domain on a uniform grid. Instead, it chooses to cluster its data samples typically around $K + 1$ locations. Figure 6 shows the first 50 input locations the active learner selects as K varies from 5 to 9, for a fixed data noise level of $\sigma_s = 0.1$. One possible explanation for this clustering behavior is that it takes only $K + 1$ data points to recover a K^{th} degree target polynomial in the absence of noise.

Second, as the data noise level σ_s increases, although the number of data clusters remains fixed, the clusters tend to be distributed away from the input origin. Figure 7 displays the first 50 input locations the active strategy selects for a 9^{th} degree polynomial approximation function class, as σ_s increases from 0.1 to 5.0. One can explain the observed behavior as follows: For higher noise levels, there is less pressure on the active learner to fit the data closely. Consequently, the prior assumption favoring polynomials with small coefficients dominates. For such "lower order" polynomials, one gets better "leverage" from data by sampling away from the origin. In the extreme case of linear regression, one gets best "leverage" by sampling data at the extreme ends of the input space.

4.3 Gaussian Radial Basis Functions

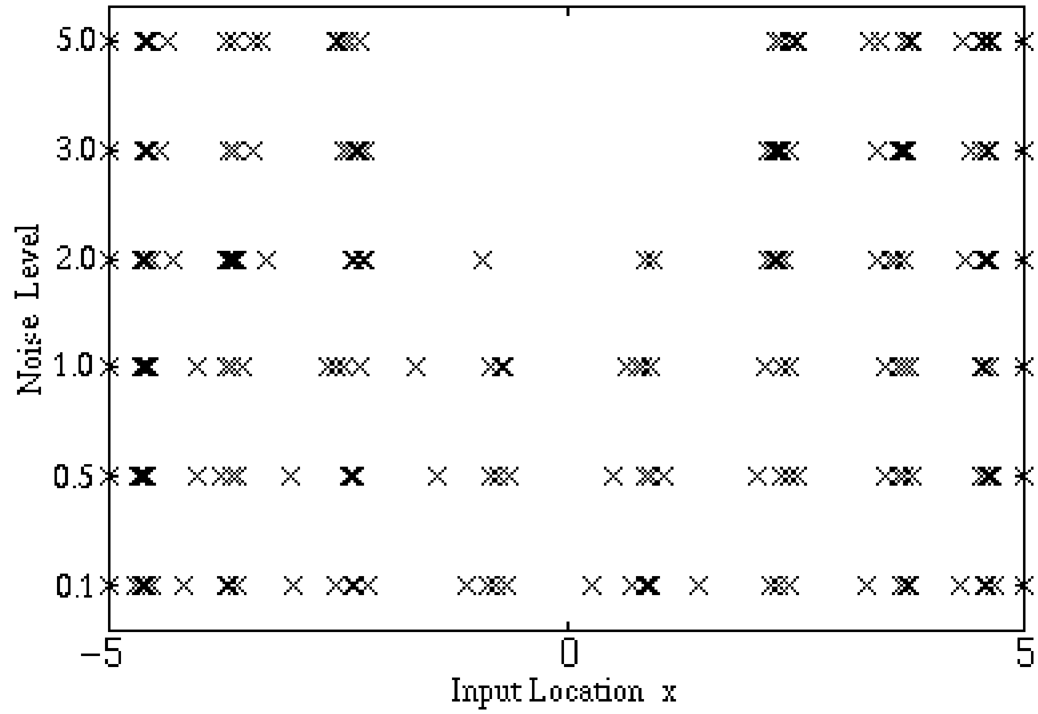


Figure 7: Distribution of the first 50 data points the active learner selects for a 9$^{\text{th}}$ degree polynomial approximation function class (i.e. $K = 9$), as the assumed data output noise level σ_s varies from 0.1 to 5.0. At higher noise levels, there is less pressure for the active learner to fit the data closely, and so it favors polynomials with small coefficients. For such "lower order" polynomials, one gets better "leverage" from data by sampling away from the origin.

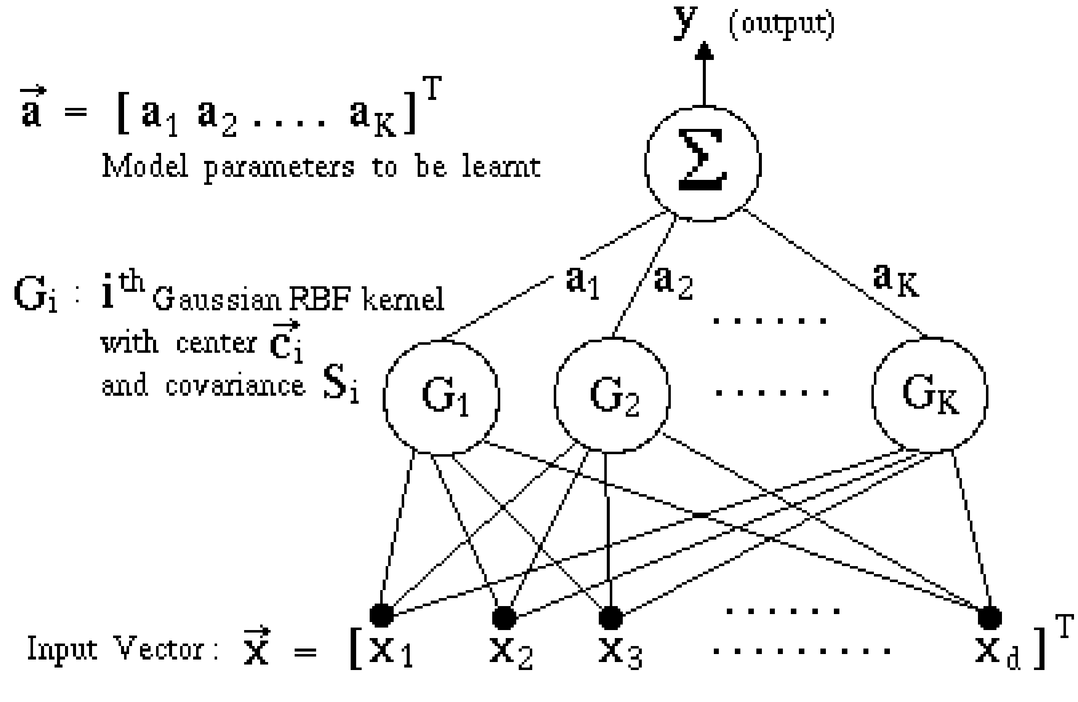


Figure 8: Gaussian radial basis function (RBF) model with K fixed centers. The learner's task is to recover the weight coefficients $\vec{a} = [a_1\ a_2\ \cdots\ a_K]^{\mathrm{T}}$.

Our final example looks at an approximation function class $\mathcal{F}$ of d-dimensional Gaussian radial basis functions with K fixed centers. Let $\mathcal{G}_i$ be the i^{th} basis function with a fixed center $\vec{c_i}$ and a fixed covariance $\mathcal{S}_i$. The model parameters to be learnt are the weight coefficients denoted by $\vec{a} = [a_1\ a_2\ \cdots\ a_K]^{\mathrm{T}}$. An arbitrary function $r \in \mathcal{F}$ in this class can thus be represented as:

$$\begin{aligned} r(\vec{x}, \vec{a}) &= \sum_{i=1}^{K} a_i \mathcal{G}_i(\vec{x}) \\ &= \sum_{i=1}^{K} a_i \frac{1}{(2\pi)^{d/2} |\mathcal{S}_i|^{1/2}} \exp(-\frac{1}{2}(\vec{x} - \vec{c_i})^{\mathrm{T}} \mathcal{S}_i^{-1} (\vec{x} - \vec{c_i})) \end{aligned}$$

We impose a prior $\mathcal{P}_{\mathcal{F}}()$ on the approximation function class $\mathcal{F}$ by putting a zero-centered Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$. Thus, for an arbitrary function $r(\cdot, \vec{a})$:

$$\mathcal{P}_{\mathcal{F}}(r(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{K/2} |\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2} \vec{a}^{\mathrm{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a}).$$

Lastly, the learner has access to noisy data of the form $\mathcal{D}_n = \{(\vec{x_i}, y_i = g(\vec{x_i}) + \eta) : i = 1, \ldots, n\}$, where g is an unknown target function and η is a zero-mean additive Gaussian noise term with variance σ_s^2. Thus for every candidate approximation function $r(\cdot, \vec{a}) \in \mathcal{F}$, $\mathcal{P}(\mathcal{D}_n | r(\cdot, \vec{a}))$ has the form:

$$\begin{aligned} \mathcal{P}(\mathcal{D}_n | r(\cdot, \vec{a})) &\propto \exp\left(-\frac{1}{2\sigma_s^2} \sum_{i=1}^{n} (y_i - r(\vec{x_i}, \vec{a}))^2\right) \\ &= \exp\left(-\frac{1}{2\sigma_s^2} \sum_{i=1}^{n} (y_i - \sum_{t=1}^{K} a_t \frac{\exp\left(-\frac{1}{2}(\vec{x_i} - \vec{c_t})^{\mathrm{T}} \mathcal{S}_t^{-1} (\vec{x_i} - \vec{c_t})\right)}{(2\pi)^{d/2} |\mathcal{S}_t|^{1/2}})^2\right) \\ &= \exp\left(-\frac{1}{2\sigma_s^2} \sum_{i=1}^{n} (y_i - \sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_i}))^2\right) \end{aligned}$$

Given a set of n data points $\mathcal{D}_n$, one can obtain a maximum a-posteriori (MAP) solution to the learning problem by finding a set of model parameters $\hat{\vec{a}}$ that maximizes $\mathcal{P}(r(\cdot, \vec{a}) | \mathcal{D}_n) = \mathcal{P}_{\mathcal{F}}(r(\cdot, \vec{a})) \mathcal{P}(\mathcal{D}_n | r(\cdot, \vec{a}))$. Let:

$$\bar{\mathbf{z}}_{\mathbf{i}} = [\mathcal{G}_1(\vec{x_i})\ \mathcal{G}_2(\vec{x_i})\ \ldots\ \mathcal{G}_K(\vec{x_i})]^{\mathrm{T}}$$

be a vector of RBF kernel output values for the i^{th} input value. One can show (see Appendix A.2.1), as in the polynomial case, that the a-posteriori RBF approximation function class distribution $\mathcal{P}(r(\cdot, \vec{a}) | \mathcal{D}_n)$ is a multivariate Gaussian centered at $\hat{\vec{a}}$ with covariance Σ_n, where:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2} \sum_{i=1}^{n} (\bar{\mathbf{z}}_{\mathbf{i}} \bar{\mathbf{z}}_{\mathbf{i}}^{\mathrm{T}}) \tag{15}$$

$$\hat{\vec{a}} = \Sigma_n \left(\frac{1}{\sigma_s^2} \sum_{i=1}^{n} \bar{\mathbf{z}}_{\mathbf{i}} y_i\right) \tag{16}$$

Notice that $\hat{\vec{a}}$ of Equation 16 is also the MAP solution the learner proposes on the basis of the data set $\mathcal{D}_n$, *regardless* of how the data points are selected. We now describe an active strategy for selecting the data optimally.

4.3.1 The Active Strategy

Recall that our goal is to derive an analytical expression for $\mathcal{U}(\hat{g_{n+1}}|D_n, \vec{x_{n+1}})$ in Equation 7, the *total output uncertainty* cost function to minimize that yields the optimal location for sampling next. As before, we go through the general active learning procedure outlined in Section 3.4 to derive an exact expression for $\hat{x_{n+1}}$, the optimal next query point.

The first derivation step is to obtain an analytical expression for $\mathcal{P}(\vec{a}|\mathcal{D}_n)$, the a-posteriori RBF approximation function class distribution. This is exactly $\mathcal{P}(r(\cdot, \vec{a})|\mathcal{D}_n)$ which we introduced in the series of equations above leading to Equation 16.

Deriving the RBF *total output uncertainty* cost function $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ requires several steps (see Appendix A.2.2 and A.2.3). We eventually get:

$$\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}}) \propto |\Sigma_{n+1}|. \tag{17}$$

Σ_{n+1} has the same form as Σ_n in Equation 16 and depends on the previous data sample locations $\{\vec{x_i} : i = 1, \ldots, n\}$, the model priors $\Sigma_{\mathcal{F}}$, the data noise variance σ_s^2, and the next sample location $\vec{x_{n+1}}$. When minimized over $\vec{x_{n+1}}$, we get $\hat{\vec{x_{n+1}}}$ as the maximum utility location where the active learner should next sample the unknown target function.

Like the polynomial class example, our RBF approximation function class $\mathcal{F}$ is also linear in its model parameters. As such, the optimal new sample location $\hat{\vec{x_{n+1}}}$ does not depend on the y data values in $\mathcal{D}_n$, but only on the previously sampled $\vec{x}$ values.

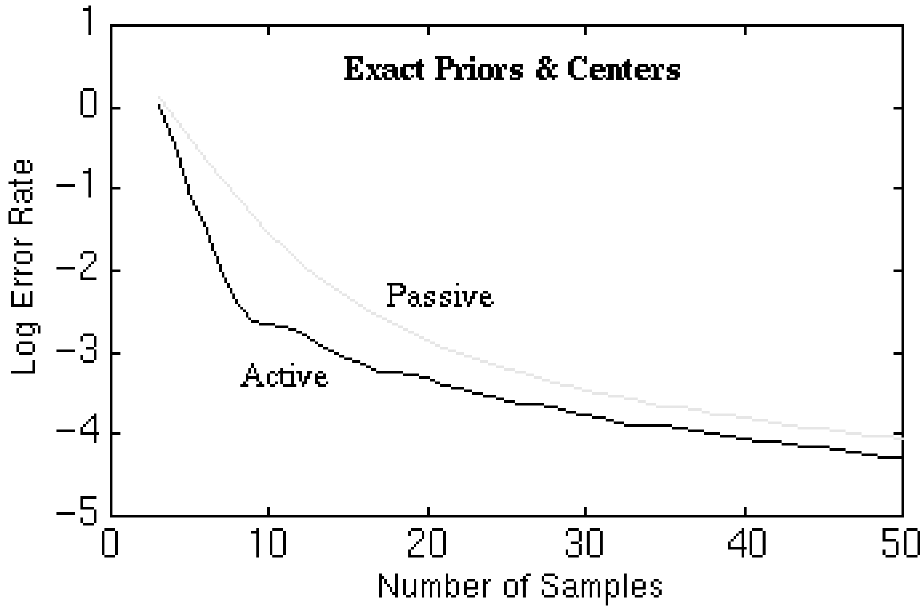
4.3.2 Simulations — Error Rate versus Number of Examples

Does the active strategy for our RBF approximation function class take fewer examples to learn an unknown target than a passive learner that draws random samples according to a uniform distribution on the input domain? We compare sample complexities for the active and passive learners under the following two conditions:

1. **The approximation and target function classes have identical priors.** For simplicity, we perform our simulations in a one-dimensional input domain $[x_{\mathtt{LO}}, x_{\mathtt{HI}}] = [-5, 5]$. The approximation and target function classes are RBF networks with $K = 8$ fixed centers, arbitrarily located within the input domain. Each RBF kernel has a fixed 1-dimensional Gaussian "covariance" of $\mathcal{S}_i = 1.0$. Finally, we assume identical independent Gaussian priors on the model parameters $\vec{a}$, i.e. $\Sigma_{\mathcal{F}} = \mathbf{I}_K = \mathbf{I}_8$, where $\mathbf{I}_K$ stands for a $K \times K$ *identity* covariance matrix.

2. **The approximation and target function classes have slightly different priors.** We use a similar RBF approximation function class with $K = 8$ fixed centers and a similar 1-dimensional Gaussian kernel "covariances" of $\mathcal{S}_i = 1.0$ for the centers. Each center is slightly displaced from its true location (i.e. its location in the target function class) by a random distance with Gaussian standard deviation $\sigma = 0.1$. The learner's priors on model parameters ($\Sigma_{\mathcal{F}} = 0.9\mathbf{I}_8$) are also slightly different from that of the target class ($\Sigma_{\mathcal{F}} = \mathbf{I}_8$).

The two simulations proceed as follows: We randomly generate 5000 target RBF functions according to the *target* model priors described above. For each target function, we collect data sets with noisy y values ($\sigma_s = 0.1$) ranging from 3 to 50 samples in size, using both the active



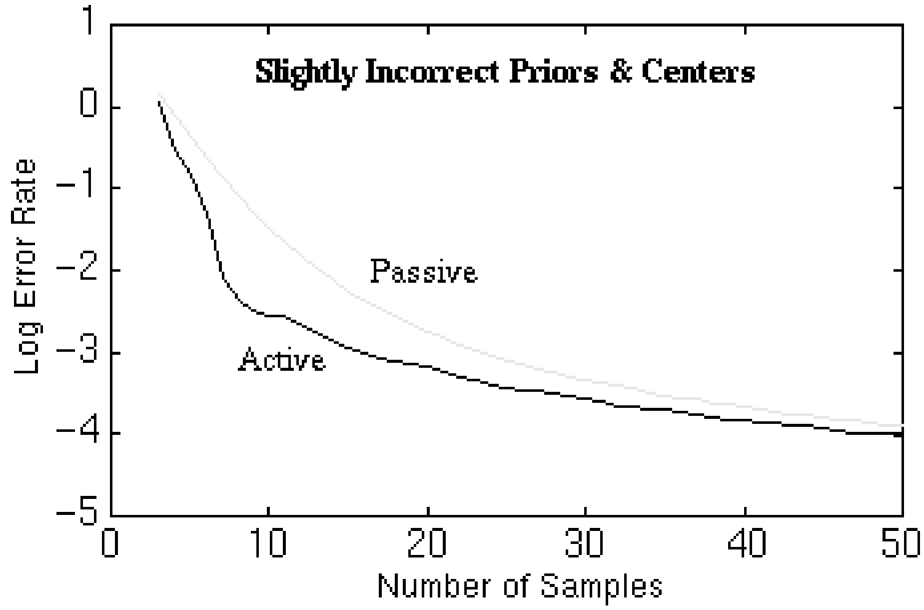


Figure 9: Comparing active and passive learning average error rates for Gaussian RBF approximators with $K = 8$ centers. **Top graph:** We use the same priors on the target and approximation function classes. **Bottom graph:** The target and approximation function classes have slightly different center locations and priors on model parameters. In both cases, the active learner has a lower average error rate than the passive learner for the same number of data points.

and passive sampling strategies. We then obtain a regularized estimate of the target function for each data set using Equation 16, and finally, we compute the actual integrated squared difference between the target and its estimate as an approximation error measure. The graphs in Figure 9 plot the average error rates (over the 5000 different target functions) for both the active and passive learners as a function of data sample size. The upper graph shows the learner using exact priors, i.e, that of the target class, while the lower graph is for the case of slightly *incorrect priors.* In both cases, the active learner has a lower average error rate than the passive learner for the same number of data points. This is especially true for small data sets.

5 Sufficiency Conditions for Pre-Computing a Data Sampling Sequence

It is noteworthy that both for learning RBF weights as well as polynomial coefficients, the new optimal sample location, $\hat{\mathbf{x}_{n+1}}$, does not depend on the y_i data values previously observed but only on the $\mathbf{x}_i$ values sampled. Thus, if the learner were to collect n data points, it can pre-compute the exact sequence of n points at which to sample from the start, even before receiving any data from the target function. This behavior has been observed by MacKay [10] for an active example selection strategy that minimizes only a model parameter variance cost function. For such cost functions, any class of approximation functions that are linear in their model parameters would exhibit such behavior.

In our framework, we minimize an output uncertainty cost function that includes both bias and variance terms. The following theorem provides sufficiency conditions on the learning problem for which our active learning formulation leads to a data selection strategy that does not depend on previously observed y_i data values.

Theorem 2 *Suppose $\mathcal{F}$ is a class of real-valued functions parameterized by $\mathbf{a} \in \Re^k$. On the basis of a data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$, let the MAP solution to the learning problem be given by $\hat{\mathbf{a}} = \arg\min_{\mathbf{a}\in\Re^k} P(g(\mathbf{a}|\mathcal{D}_n))$. Then the following three conditions guarantee that the choice of $\hat{\mathbf{x}}_{n+1}$ will be independent of the previously observed y_i's in $\mathcal{D}_n$.*

1. *$P(g(\mathbf{a})|D_n)$ can be expressed as $Q((\mathbf{a} - \hat{\mathbf{a}}(\mathcal{D}_n)), \{\mathbf{x}_i : i = 1 \ldots n\})$ where Q is some arbitrary function that does not depend on the data, $\mathcal{D}_n$. In other words, the y_i terms of $\mathcal{D}_n$ do not appear anywhere else in $P(g(\mathbf{a})|D_n) = Q((\mathbf{a} - \hat{\mathbf{a}}(\mathcal{D}_n)), \{\mathbf{x}_i : i = 1 \ldots n\})$ other than in $\hat{\mathbf{a}}$.*

2. *$\mathcal{F}$ is linear in its parameters $\mathbf{a}$, i.e.: $g_{\mathbf{a}_1+\mathbf{a}_2}(\mathbf{x}) = g_{\mathbf{a}_1}(\mathbf{x}) + g_{\mathbf{a}_2}(\mathbf{x})$.*

3. *The prior distribution on model parameters $\mathbf{a}$ must have support equal to $\Re^k$.*

As it turns out, if the conditions of the above theorem are met, the output uncertainty function $\mathcal{U}$ of our active learning formulation can also be analytically solved. Note that these conditions are sufficient to guarantee analytical tractability. They are by no means necessary and it would be useful to shed stronger light on the kinds of function classes and probability distributions for which exact active algorithms exist.

6 Active Example Selection and the "Boot-strap" Paradigm

Recall from Section 3.3 that our active learning strategy chooses its next sample location by minimizing the *total output uncertainty* cost function in Equation 7. For convenience, we reproduce the relevant expressions below:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{-\infty}^{\infty} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]dy_{n+1}. \quad (18)$$

where:

$$E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g},g)dg = \int_{g\in\mathcal{F}} \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)\delta(\hat{g},g)dg.$$

and:

$$\mathcal{P}(y_{n+1}|\vec{x_{n+1}},\mathcal{D}_n) \propto \int_{f\in\mathcal{F}} P(\mathcal{D}_n \cup (\vec{x_{n+1}},y_{n+1})|f)\mathcal{P}_{\mathcal{F}}(f)df.$$

Clearly, from the three equations above, the cost function $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n,\vec{x_{n+1}})$ may not have a simple analytical form for many approximation function classes. The current active learning formulation may therefore be computationally intractable for arbitrary approximation function classes in general.

One way of making the active learning task computationally tractable is to define simpler but less "complete" cost functions for measuring the potential utility of new sample locations. To conclude this paper, we look at one such simplification approach and show how it leads to the "boot-strap" example selection strategy we used for training object and pattern detection systems. We also show empirically that even though the "boot-strap" strategy may be "sub-optimal" in its choice of new examples, it still outperforms random sampling in training a frontal face detection system. As such, we maintain that the "boot-strap" strategy is still an effective means of sifting through unmanageably large data sets that would otherwise make learning intractable.

6.1 A Simpler Example Utility Measure

Let g be an *unknown* target function that we want to estimate by means of an approximation function in $\mathcal{F}$, $\mathcal{D}_n = \{(\vec{x_i},y_i) \in \Re^d \times \Re | i = 1,\ldots,n\}$ be the set of n data points seen so far, and $\hat{g_n} \in \mathcal{F}$ be the current regularized estimate for g. Recall from Section 3.3 that our learning goal is to minimize an *expected misfit* notion between g and its regularized solution, and our optimal sampling strategy chooses the next input location $\vec{x_{n+1}} \in \Re^d$ that best minimizes $E_{\mathcal{F}}[\delta(\hat{g_{n+1}},g)|\mathcal{D}_n \cup (\vec{x_{n+1}},y_{n+1})]$, the EISD between g and its new estimate $\hat{g_{n+1}}$.

We now describe a different example selection *heuristic*, based on a simpler but less comprehensive example utility measure, that also attempts to efficiently reduce the *expected misfit* between g and $\hat{g_{n+1}}$. Let $\mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x})$ be a *local* "uncertainty" measure for the current estimate $\hat{g_n}$ at $\vec{x}$:

$$\mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x}) = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)(\hat{g_n}(\vec{x}) - g(\vec{x}))^2 dg \tag{19}$$

Notice that unlike the EISD which globally characterizes an entire approximation function, $\mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x})$ is just a *local* "error bar" measure between g and $\hat{g_n}$ only at $\vec{x}$. In a loose sense, one can view the *local* "error bar" $\mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x})$ as information that the learner *lacks* at input location $\vec{x}$ for an exact solution. Given such an interpretation, one can also regard $\mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x})$ as an example utility measure, because the learner essentially gains the information it lacks at $\vec{x}$ by sampling there. The new example selection *heuristic* can thus be formulated as choosing the next sample where the new example's *utility value* is greatest, i.e., sampling next where the learner lacks most information:

$$\vec{x_{n+1}} = \arg\max_{\vec{x}\in\Re^d} \mathcal{L}(\hat{g_n}|\mathcal{D}_n,\vec{x}) = \arg\max_{\vec{x}\in\Re^d} \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)(\hat{g_n}(\vec{x}) - g(\vec{x}))^2 dg \tag{20}$$

We stress again that the new example selection *heuristic* differs from our original active learning formulation only in the example utility cost function it uses. The new heuristic uses a simpler example utility measure, based on the *current* estimate's *local* uncertainty properties instead of the *new* estimate's expected *global* uncertainty properties. In doing so, it implicitly assumes the following:

1. The learning goal is still to minimize the *expected misfit*, i.e., the *total output uncertainty* between g and its estimate $\hat{g}$.

2. One can bring about a proportionate decrease in the *new* estimate's *global* output uncertainty level by *locally* reducing the *current* estimate's output uncertainty at some arbitrary location.

3. There is a better chance of significantly reducing the *local* output uncertainty at a point whose current uncertainty level is high. Furthermore, one can best reduce the *local* uncertainty level at a point by sampling directly at the point.

4. It follows from the previous assumptions that one can most efficiently minimize the new estimate's *global* uncertainty level by gathering new data where the current estimate's *local* output uncertainty level is highest.

Although the above assumptions appear intuitively reasonable, one should still be able to find approximation function classes that do not meet the above assumptions. This suggests that in general, the new *heuristic* may still be a "sub-optimal" sampling strategy with respect to the active learning goal of maximally reducing the *expected misfit* between g and $\hat{g}$ with each new data point. Nevertheless, Equation 19 is clearly a much simpler example utility cost function than Equation 18, which makes the new heuristic computationally tractable for a much larger range of approximation function classes.

Are there approximation function classes for which the new heuristic and the original active example selection strategy are functionally equivalent? MacKay [10] has shown that if one approximates the current a-posteriori model parameter distribution, i.e. $\mathcal{P}(\vec{a}|\mathcal{D}_n) \equiv \mathcal{P}(g(\cdot,\vec{a})|\mathcal{D}_n)$, as a multi-dimensional Gaussian probability density centered at $\hat{\vec{a}}$, the optimal model parameter estimate, then minimizing the new estimate's *global* uncertainty level reduces to sampling where the current estimate's "error bars" are greatest. MacKay has also observed that for *linear* approximation function classes (i.e., one for which $g(\vec{x},\vec{a}) = \sum_{i=1}^{K} a_i \psi_i(\vec{x})$) with quadratic penalty functions, $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ is *exactly* a multi-dimensional Gaussian probability density centered at $\hat{\vec{a}}$, which in turn suggests that the two sampling strategies are computationally equivalent for such approximation function classes. We refer the interested reader to MacKay's work [10] for further details.

6.2 Example Selection in a Real Pattern Detection Training Scenario

We now discuss a variant of the simplified example selection heuristic, used for training a frontal view human face detection system. In order to train a face detection system with finite computation resource, one must first acquire a comprehensive but tractably small database of "face" and "non-face" example patterns. For "face" patterns, one can simply collect all frontal face views from mugshot databases and other image archives, and still have a manageably small data set. For "non-face" patterns, the task is more tricky. In essence, any normalized window pattern that does not tightly contain a frontal human face is a valid "non-face" training example. Clearly, our "non-face" example set can grow intractably large if we should include every available "non-face" image patch in our training database.

Notice that our learning scenario for face detection differs slightly from the original active learning scenario presented earlier. In the original setting, one assumes that data measurements are relatively expensive or slow, and we seek the next sample location that best maximizes the expected amount of information gained. In our current scenario, we have an immense amount of available "non-face" data from which we wish to select a small training sample most useful for our learning task. The learner in our face detection scenario also has an added advantage: it already knows the actual output value (i.e., class label) of every candidate "non-face" data point even before they are selected.

Clearly, the current learning scenario reduces to the original active learning setting if we ignore output values (i.e., class labels) of the candidate data points when deciding which new patterns to select. Despite apparent differences in form, both learning scenarios address the same central issue, namely how a learner can select new examples intelligently by estimating the utility of candidate data points. In fact, we shall see shortly that one can still borrow ideas developed for the original active learning scenario to approach example selection in the face detection scenario.

6.3 The "Boot-strap" Paradigm

To constrain the number of "non-face" patterns in our training database, we introduce a "boot-strap" paradigm that incrementally selects "non-face" patterns highly relevant to the learning problem. The "boot-strap" strategy reduces the number of "non-face" patterns needed to train a highly robust face detector. We outline the idea below:

1. Start with a small and possibly highly non-representative set of "non-face" examples in the training database.

2. Train a face classifier to output a value of '1' for face examples and '0' for non-face examples using patterns from the current example database.

3. Run the trained face detector on a sequence of images with no faces. Collect all (or a random subset of) the "non-face" patterns that the current system wrongly classifies as "faces" (i.e., an output value of > 0.5). Add these "non-face" patterns to the training database as new negative examples.

4. Return to Step 2.

More generally, one can use the "boot-strap" paradigm to select useful training examples from either pattern class in an arbitrary pattern detection problem:

1. Start with a small and possibly highly non-representative example set in the training database.

2. Train a pattern classifier to output a value of '1' for positive examples and '0' for negative examples using patterns from the current example database.

3. Run the trained pattern classifier on a sequence of images. Collect all (or a random subset of) the wrongly classified patterns and add them to the training database as new correctly labeled examples.

4. Return to Step 2.

At the end of each iteration, the "boot-strap" paradigm augments the current data set with new patterns that the current system classifies wrongly. We argue that this strategy of collecting wrongly classified patterns as new training examples is reasonable, because one can expect these new examples to improve the classifier's performance by steering it away from its current mistakes.

One can reason about the "boot-strap" paradigm as a variant of the previously discussed simplified example selection heuristic. First, as in the original active learning spirit, "boot-strapping" attempts to select only high utility examples for training. During, each example selection step, if "boot-strapping" were to follow the active learning procedure *exactly*, then it should select only the highest utility example available and continue the training process. Instead, we make the "boot-strap" paradigm less restrictive by allowing the learner to select one or more "high utility" examples between training cycles. Notice that the highest utility example may not even be among those selected. Second, the simplified heuristic estimates an example's utility by computing $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$, the *local* "error bar" measure of Equation 19. In "boot-strapping", we take advantage of the already available output value (i.e., class label) at each candidate location to implement a coarse but very simple *local* "error bar" measure for selecting new examples. Points that are classified correctly have low actual output errors. We assume that these points also have low uncertainty "error bars" and so we ignore them as examples containing little new information. Conversely, points that are wrongly classified have high actual output errors. We conveniently assume that these points also have high *local* uncertainty "error bars" and are therefore "high utility" examples suitable for the learning task. We stress again that we are assuming actual output errors and *local* uncertainty "error bars" are highly correlated measurements, which may not always be a valid assumption.

6.4 Sample Complexity of "Boot-strapping"

Does the "boot-strap" strategy yield better classification results with fewer training examples than a passive learner that receives randomly drawn patterns? We show empirically that this is indeed the case for our frontal face detection learning scenario. Using a Radial Basis Function-like network architecture for pattern classification, we trained a frontal view face detection system (see [20] for details). This was done in two "boot-strap" cycles. The final training database contains 4150 "face" patterns and over 40000 "non-face" patterns, of which about 6000 were selected during the second "boot-strap" cycle. As a control, we trained a second system without "boot-strapping". The second training database contains the same 4150 "face" patterns and another 40000 randomly selected "non-face" patterns. We used the same "face" and "non-face" Gaussian clusters from the first system to model the target and near-miss pattern distributions in the second system.

We ran both systems on a test database of 23 cluttered images with 147 frontal faces. The first system missed 23 faces and produced 13 false detects, while the second system had 15 missed faces and 44 false alarms. Notice that the first system trained with "boot-strapping" has a lower *total* mis-classification rate than the control trained without "boot-strapping". The "boot-strap" system misses more faces but has a much smaller false alarm rate for non-faces. This is because we have used "boot-strapping" to only select better "non-face" patterns while leaving the total number of "face" patterns unchanged. Consequently, the system is better able to reject non-face patterns at a slight expense of correctly detecting faces.

Our comparison suggests that even though the "boot-strap" strategy may be "sub-optimal" in choosing new training patterns, it is still a very simple and effective technique for sifting through unmanageably large data sets for useful examples.

7 Conclusion

We have developed a framework for choosing examples usefully and investigated its plausibility as a reasonable paradigm for active learning. This framework rests on techniques from Optimal Experiment Design, generalizes the work of MacKay in a Bayesian setting and is similar to the work of Sollich in a non-Bayesian one. We then show how precise algorithms can be derived for certain function classes and explore the improved perfomance of these algorithms.

There are a number of useful directions to explore further. First, it is useful to understand what sort of function classes would yield tractable solutions to the objective function criteria set up in our framework. We have attempted a partial answer to this, but the analysis is by no means complete. Another useful direction is to consider forms of activity other than example selection

Figure 10: Comparing face detection results for two systems: one trained without "boot-strapping" and the other with "boot-strapping". The system trained without "boot-strapping" does a poorer job at discarding non-face patterns. **Top pair:** The system without "boot-strapping" finds the frontal face correctly but also makes four false detects — three near the top-left image corner and one near Mia Hamm's left knee. **Bottom pair:** The system without "boot-strapping" makes two false detects in the complex background, left of Iolaus' face.

that might be useful in practice. We have made no attempt to discuss this issue in the current paper.

There have been some previous attempts to develop theoretical frameworks and analyses of active learning situations. It is crucial that these frameworks eventually translate into some sort of working algorithms that can be used to solve complex learning problems. As an attempt to bridge the gap between theory and practice, we have argued how the bootstrap paradigm can be derived from our framework and implemented with some success in a sophisticated face detection system.

A The Active Learning Procedure

This appendix derives the active example selection procedures for polynomial approximators and Gaussian radial basis functions. Specifically, we show the steps leading to Equations 11 and 17, i.e., the *total output uncertainty* cost functions $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ for polynomial approximators and Gaussian RBFs respectively.

A.1 Polynomial Approximators

Let $\mathcal{F}$ be a univariate polynomial approximation function concept class with maximum degree K:

$$\mathcal{F} = \{g(x, \vec{a}) = g(x, a_0, \ldots, a_K) = \sum_{i=0}^{K} a_i x^i\}.$$

The model parameters to be learnt are $\vec{a} = [a_0\ a_1\ \ldots\ a_K]^{\mathtt{T}}$ and $x \in [x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ is the input variable . The prior distribution on $\mathcal{F}$ is a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$:

$$\mathcal{P}_{\mathcal{F}}(g(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathtt{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}). \tag{21}$$

Our task is to approximate an *unknown* target function $g \in \mathcal{F}$ within the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ on the basis of *noisy* sampled data: $\mathcal{D}_n = \{(x_i, y_i = g(x_i) + \eta)|i = 1, \ldots, n\}$, where η is an additive zero-mean Gaussian noise term with variance σ_s^2.

A.1.1 The A-Posteriori Function Class Distribution

We first derive the a-posteriori distribution on function class $\mathcal{F}$ given data $\mathcal{D}_n$, i.e., $\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a})$. Since $\mathcal{D}_n$ is sampled under additive zero-mean Gaussian noise with variance σ_s^2, we have:

$$\begin{aligned} \mathcal{P}(\mathcal{D}_n|\vec{a}) &\propto \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - g(x_j, \vec{a}))^2\right) \\ &= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=0}^{K} a_t x_j^t)^2\right) \end{aligned} \tag{22}$$

Let $\bar{\mathbf{x}}_{\mathbf{j}} = [1\ x_j\ x_j^2\ \ldots\ x_j^K]^{\mathtt{T}}$ be a power vector of the $j^{\mathtt{th}}$ input value. One can expand the exponent term in Equation 22 as follows:

$$\begin{aligned}\left(y_j - \sum_{t=0}^{K} a_t x_j^t\right)^2 &= y_j^2 + \left(\sum_{t=0}^{K} a_t x_j^t\right)^2 - 2y_i \sum_{t=0}^{K} a_t x_j^t \\ &= y_j^2 + \vec{a}^{\mathrm{T}}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - y_j\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}}\vec{a} - \vec{a}^{\mathrm{T}}\bar{\mathbf{x}}_{\mathbf{j}}y_j\end{aligned}$$

So:

$$\begin{aligned}\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\left(y_j - \sum_{t=0}^{K} a_t x_j^t\right)^2 &= \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a} \\ &\quad - (\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j)\end{aligned}$$

The polynomial prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is given in Equation 21. The a-posteriori distribution is thus:

$$\begin{aligned}\mathcal{P}(\vec{a}|\mathcal{D}_n) &\propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a}) \\ &\propto \exp\left(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}\right)\exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=0}^{K} a_t x_j^t)^2\right) \\ &= \exp\left[-\frac{1}{2}\left(\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a}\right.\right. \\ &\qquad \left.\left. -(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j)\right)\right] \\ &= \exp\left[-\frac{1}{2}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a}\right.\right. \\ &\qquad \left.\left. -(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j)\right)\right] \qquad (23)\end{aligned}$$

Completing the square in Equation 23 yields:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}}) - \hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2\right)\right] \qquad (24)$$

where:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}}) \qquad (25)$$

$$\hat{\vec{a}} = \Sigma_n(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j) \qquad (26)$$

Notice that neither of the two terms $\hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}}$ and $\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2$ in Equation 24 depend on the polynomial model parameters $\vec{a}$. This means that we can rewrite Equation 24 as:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \tag{27}$$

Clearly, Equation 27 is multivariate Gaussian in form. To express $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ as a standard probability distribution on $\vec{a}$ that integrates to 1, we simply introduce into Equation 27 the appropriate normalizing constants:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n|^{1/2}}\exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \tag{28}$$

Thus, the polynomial a-posteriori function class distribution is a multivariate Gaussian centered at $\hat{\vec{a}}$ (Equation 26) with covariance Σ_n (Equation 25).

A.1.2 The Polynomial EISD Measure

Recall from Section 3.3 that the *total output uncertainty* cost functions $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ is simply the *expected* EISD measure between g and its *new* estimate $\hat{g_{n+1}}$, if the learner samples next at $\vec{x_{n+1}}$. We now derive an expression for the EISD between g and its *current* estimate $\hat{g_n}$ given $\mathcal{D}_n$. We shall use this result later to derive an expression for $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$.

The *expected integrated squared difference* (EISD) between an unknown target g and its estimate $\hat{g}$ given $\mathcal{D}_n$ is:

$$E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g},g)dg$$

where $\delta(\hat{g},g)$ is a standard integrated squared difference measure between two functions over the input space $\Re^d$ or some appropriate region of interest:

$$\delta(\hat{g},g) = \int_{\vec{x}\in\Re^d}(g(\vec{x})-\hat{g}(\vec{x}))^2 d\vec{x}.$$

For our polynomial approximation function class, the *optimal* estimate for g given $\mathcal{D}_n$ has model parameters $\hat{\vec{a}}$ (Equation 26), since this is where $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ has a global maximum. Let $\hat{\vec{a}} = [\hat{a_0}\ \hat{a_1}\ \ldots\ \hat{a_K}]^{\mathrm{T}}$ and $\bar{\mathbf{x}} = [1\ x\ x^2\ \ldots\ x^K]^{\mathrm{T}}$, one can rewrite $\delta(\hat{g},g)$ in terms of polynomial model parameters as:

$$\begin{aligned}
\delta(\hat{\vec{a}},\vec{a}) &= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}[g(x,\vec{a})-g(x,\hat{\vec{a}})]^2dx = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\left[(\sum_{i=0}^{K}a_ix^i)-(\sum_{i=0}^{K}\hat{a}_ix^i)\right]^2 dx \\
&= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\left[\sum_{i=0}^{K}(a_i-\hat{a}_i)x^i\right]^2 dx = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}(\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\bar{\mathbf{x}}\bar{\mathbf{x}}^{\mathrm{T}}(\vec{a}-\hat{\vec{a}})dx \\
&= (\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\left(\int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\bar{\mathbf{x}}\bar{\mathbf{x}}^{\mathrm{T}}dx\right)(\vec{a}-\hat{\vec{a}}) \\
&= (\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}})
\end{aligned} \tag{29}$$

where $\mathbf{A}$ is a constant $(K+1)\times(K+1)$ matrix of numbers whose $(i,j)^{\mathtt{th}}$ element is:

$$\mathbf{A}(i,j) = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} x^{(i+j-2)} dx$$

Substituting Equations 28 and 29 into the EISD expression, we get:

$$\begin{aligned} E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] = \int_{\vec{a}\in\Re^{K+1}} \mathcal{P}(\vec{a}|\mathcal{D}_n)\delta(\hat{\vec{a}},\vec{a})d\vec{a} \\ &= \int_{\vec{a}\in\Re^{K+1}} \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n|^{1/2}} \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \\ &\qquad (\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}})\, d\vec{a} \end{aligned} \tag{30}$$

Making the following change of variables: $\vec{q} = \mathbf{A}^{\frac{1}{2}}(\vec{a}-\hat{\vec{a}})$, and noting that the integration bounds on $\vec{q}$ is still $\Re^{K+1}$, Equation 30 becomes:

$$\begin{aligned} E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] \\ &= \int_{\vec{q}\in\Re^{K+1}} \frac{1}{(2\pi)^{(K+1)/2}|\mathbf{A}|^{1/4}|\Sigma_n|^{1/2}|\mathbf{A}|^{1/4}} \\ &\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathtt{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathtt{T}}\vec{q}\, d\vec{q} \\ &= \int_{\vec{q}\in\Re^{K+1}} \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n\mathbf{A}|^{1/2}} \\ &\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathtt{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathtt{T}}\vec{q}\, d\vec{q} \\ &= |\Sigma_n\mathbf{A}| \quad \propto \quad |\Sigma_n| \end{aligned} \tag{31}$$

since $\mathbf{A}$ is just a constant matrix of numbers.

Notice from Equation 25 that Σ_n depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the output noise variance σ_s^2 and the previously sampled input locations $\{x_1,x_2,\ldots,x_n\}$. It does not depend on the previous y data values actually observed. In other words, the previously observed y data values in $\mathcal{D}_n$ *do not* affect the EISD measure (Equation 31) for this polynomial function concept class!

A.1.3 The Total Output Uncertainty Measure

The *total output uncertainty* cost function resulting from sampling next at $\vec{x_{n+1}}$ is given by Equation 7. We rewrite the expression below in terms of our polynomial model parameters:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n,x_{n+1}) = \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|x_{n+1},\mathcal{D}_n)E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n\cup(x_{n+1},y_{n+1})]\, dy_{n+1}. \tag{32}$$

where:

$$\mathcal{P}(y_{n+1}|x_{n+1},\mathcal{D}_n) \propto \int_{\vec{a}\in\Re^{K+1}} \mathcal{P}(\mathcal{D}_n \cup (x_{n+1},y_{n+1})|\vec{a})\mathcal{P}_{\mathcal{F}}(\vec{a})\, d\vec{a}.$$

It is clear from Equation 32 that $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, x_{n+1})$ is merely the *expected* EISD value between g and its *new* estimate, weighted and averaged over all possible values of y_{n+1} at x_{n+1}. Recall from Equation 25 however, that for this polynomial function class, the EISD between g and its estimate $\hat{g}$ depends only on the input x_i values in $\mathcal{D}_n$ and not on the observed y_i values. This means that $E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n \cup (x_{n+1},y_{n+1})]$, the new EISD resulting from sampling next at x_{n+1}, does not depend on y_{n+1}! Equation 32 can therefore be further simplified, which leads to the following closed form expression for the *total output uncertainty* cost function, given also in Equation 11:

$$\begin{aligned}
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, x_{n+1}) &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n \cup (x_{n+1},y_{n+1})] \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|x_{n+1},\mathcal{D}_n)\, dy_{n+1} \\
&= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n \cup (x_{n+1},y_{n+1})] \\
&= |\Sigma_{n+1}\mathbf{A}| \quad \propto \quad |\Sigma_{n+1}|
\end{aligned} \tag{33}$$

Here, Σ_{n+1} has exactly the same form as Σ_n in Equation 25, and depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the output noise variance σ_s^2 and the data input locations $\{x_1, x_2, \ldots, x_n, x_{n+1}\}$.

A.1.4 The Polynomial Smoothness Prior

For our polynomial function class, we have assumed the following multi-dimensional Gaussian prior distribution on model parameters $\vec{a} = [a_0\ a_1\ \ldots\ a_K]^{\mathrm{T}}$:

$$\mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}),$$

where $\Sigma_{\mathcal{F}}$ is a $(K+1)\times(K+1)$ covariance matrix. If one assumes an *independent* Gaussian distribution with variance σ_i^2 on each parameter a_i, then $\Sigma_{\mathcal{F}}$ is simply a diagonal matrix with the independent variance terms $\{\sigma_i^2|i=0,\ldots,K\}$ on its principal diagonal.

Let $p \in \mathcal{F}$ be a polynomial function in the approximation concept class. We consider here a slightly different prior distribution on $\mathcal{F}$ based on a global "smoothness" measure:

$$\begin{aligned}
\mathcal{P}_{\mathcal{F}}(\vec{a}) &\propto \exp(-\mathcal{Q}(p(\cdot,\vec{a})))\exp(-\frac{a_0^2}{2\sigma_0^2}) \\
&= \exp\left[-\frac{1}{2}\left(2\mathcal{Q}(p(\cdot,\vec{a})) + \frac{a_0^2}{\sigma_0^2}\right)\right]
\end{aligned} \tag{34}$$

where the "smoothness" term is:

$$\mathcal{Q}(p(\cdot,\vec{a})) = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} \left[\frac{dp(x,\vec{a})}{dx}\right]^2\, dx.$$

We shall show that despite the apparent difference in general form, $\mathcal{P}_{\mathcal{F}}(\vec{a})$ still simplifies to a multi-dimensional Gaussian distribution, whose new covariance term $\Sigma_{\mathcal{F}}$ is given by Equation 14. First we express $\mathcal{Q}(p(\cdot,\vec{a}))$ in terms of the polynomial model parameters $\vec{a}$:

$$
\begin{aligned}
p(x) &= \sum_{t=0}^{K} a_t x^t \\
\frac{d\,p(x)}{dx} &= \sum_{t=1}^{K} a_t t x^{t-1} \\
\left(\frac{d\,p(x)}{dx}\right)^2 &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_K \end{bmatrix}^{\mathtt{T}} \begin{bmatrix} 1 & 2x & 3x^2 & \cdots & Kx^{K-1} \\ 2x & 4x^2 & 6x^3 & & 2Kx^K \\ 3x^2 & 6x^3 & 9x^4 & & \vdots \\ \vdots & & & \ddots & \\ Kx^{K-1} & 2Kx^K & \cdots & & K^2x^{2K-2} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_K \end{bmatrix} \\
\mathcal{Q}(p(\cdot,\vec{a})) &= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} \left(\frac{d\,p(x)}{dx}\right)^2 dx \quad = \quad [a_1\ a_2\ \ldots\ a_K]\,\mathbf{N}\,[a_1\ a_2\ \ldots\ a_K]^{\mathtt{T}}
\end{aligned}
$$

$\mathbf{N}$ is a constant $K \times K$ matrix of numbers whose $(i,j)^{\mathtt{th}}$ element is:

$$\mathbf{N}(i,j) = \frac{ij}{i+j-1}(x_{\mathtt{HI}}^{i+j-1} - x_{\mathtt{LO}}^{i+j-1})$$

Next, we substitute the above result into the exponent of Equation 34. We get:

$$
\begin{aligned}
2\mathcal{Q}(p(\cdot,\vec{a})) + \frac{a_0^2}{\sigma_0^2} &= [a_0\ a_1\ \ldots\ a_K] \begin{bmatrix} 1/\sigma_0^2 & \vec{0}^{\mathtt{T}} \\ \vec{0} & 2\mathbf{N} \end{bmatrix} [a_0\ a_1\ \ldots\ a_K]^{\mathtt{T}} && (35) \\
&= \vec{a}^{\mathtt{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a} \\
\mathcal{P}_{\mathcal{F}}(\vec{a}) &\propto \exp\left[-\frac{1}{2}\left(2\mathcal{Q}(p(\cdot,\vec{a})) + \frac{a_0^2}{\sigma_0^2}\right)\right] \\
&= \exp\left[-\frac{1}{2}\vec{a}^{\mathtt{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}\right] && (36)
\end{aligned}
$$

where the $K \times K$ matrix $\mathbf{N}$ is as defined above. Thus, from Equation 36, we see that the new prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is indeed multi-dimensional Gaussian in form with covariance $\Sigma_{\mathcal{F}}$ as given below and in Equation 14:

$$\Sigma_{\mathcal{F}}^{-1}(i,j) = \begin{cases} 1/\sigma_0^2 & \text{if } i = j = 1 \\ 2\frac{(i-1)(j-1)}{i+j-3}(x_{\mathtt{HI}}^{i+j-3} - x_{\mathtt{LO}}^{i+j-3}) & \text{if } 2 \le i \le K+1 \text{ and } 2 \le j \le K+1 \\ 0 & \text{otherwise} \end{cases} \qquad (37)$$

A.2 Gaussian Radial Basis Functions

Our next example is a d-dimensional Gaussian Radial Basis Function class $\mathcal{F}$ with K fixed centers. The analysis here is very similar to the polynomial case presented in the previous section. Let $\mathcal{G}_i$ be the i^{th} basis function with a fixed center $\vec{c_i}$ and a fixed covariance $\mathcal{S}_i$. The model parameters to be learnt are the RBF weight coefficients denoted by $\vec{a} = [a_1\ a_2\ \ldots\ a_K]^{\mathrm{T}}$. An arbitrary function $r \in \mathcal{F}$ in this class can be represented as:

$$\begin{aligned} r(\vec{x}, \vec{a}) &= \sum_{i=1}^{K} a_i \mathcal{G}_i(\vec{x}) \\ &= \sum_{i=1}^{K} a_i \frac{1}{(2\pi)^{d/2} |\mathcal{S}_i|^{1/2}} \exp(-\frac{1}{2}(\vec{x} - \vec{c_i})^{\mathrm{T}} \mathcal{S}_i^{-1} (\vec{x} - \vec{c_i})) \end{aligned}$$

The prior distribution on $\mathcal{F}$ is a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters:

$$\mathcal{P}_{\mathcal{F}}(r(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{K/2} |\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2} \vec{a}^{\mathrm{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a}). \tag{38}$$

Lastly, the learner has access to noisy data of the form $\mathcal{D}_n = \{(\vec{x_i}, y_i = g(\vec{x_i}) + \eta) : i = 1, \ldots, n\}$, where $g \in \mathcal{F}$ is an unknown target function and η is a zero-mean additive Gaussian noise term with variance σ_s^2. The learning task is to recover g from $\mathcal{D}_n$.

A.2.1 The A-Posteriori Function Class Distribution

We first derive the a-posteriori distribution on function class $\mathcal{F}$ given data $\mathcal{D}_n$, i.e., $\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(\vec{a}) \mathcal{P}(\mathcal{D}_n|\vec{a})$. Since $\mathcal{D}_n$ is sampled under additive zero-mean Gaussian noise with variance σ_s^2, we have:

$$\begin{aligned} \mathcal{P}(\mathcal{D}_n|\vec{a}) &\propto \exp\left(-\frac{1}{2\sigma_s^2} \sum_{j=1}^{n} (y_j - r(\vec{x_j}, \vec{a}))^2 \right) \\ &= \exp\left(-\frac{1}{2\sigma_s^2} \sum_{j=1}^{n} (y_j - \sum_{t=1}^{K} a_t \frac{\exp\left(-\frac{1}{2}(\vec{x_j} - \vec{c_t})^{\mathrm{T}} \mathcal{S}_t^{-1} (\vec{x_j} - \vec{c_t})\right)}{(2\pi)^{d/2} |\mathcal{S}_t|^{1/2}})^2 \right) \\ &= \exp\left(-\frac{1}{2\sigma_s^2} \sum_{j=1}^{n} (y_j - \sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_j}))^2 \right) \end{aligned} \tag{39}$$

Let $\bar{\mathbf{z_j}} = [\mathcal{G}_1(\vec{x_j})\ \mathcal{G}_2(\vec{x_j})\ \ldots\ \mathcal{G}_K(\vec{x_j})]^{\mathrm{T}}$ be a vector of kernel output values for the j^{th} input value. One can expand the exponent term in Equation 39 as follows:

$$\begin{aligned} (y_j - \sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_j}))^2 &= y_j^2 + \left(\sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_j}) \right)^2 - 2y_j \sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_j}) \\ &= y_j^2 + \vec{a}^{\mathrm{T}} (\bar{\mathbf{z_j}} \bar{\mathbf{z_j}}^{\mathrm{T}}) \vec{a} - y_j \bar{\mathbf{z_j}}^{\mathrm{T}} \vec{a} - \vec{a}^{\mathrm{T}} \bar{\mathbf{z_j}} y_j \end{aligned}$$

So:

$$\begin{aligned}
\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\left(y_j-\sum_{t=1}^{K}a_t\mathcal{G}_t(\vec{x_j})\right)^2 &= \frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j^2+\vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a} \\
&\quad -(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\vec{a}-\vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j)
\end{aligned}$$

The Gaussian RBF prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is as given in Equation 38. The a-posteriori distribution is thus:

$$\begin{aligned}
\mathcal{P}(\vec{a}|\mathcal{D}_n) &\propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a}) \\
&\propto \exp\left(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}\right)\exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j-\sum_{t=1}^{K}a_t\mathcal{G}_t(\vec{x_j}))^2\right) \\
&= \exp\left[-\frac{1}{2}\left(\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}+\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j^2+\vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a}\right.\right. \\
&\qquad \left.\left. -\ (\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\vec{a}-\vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j)\right)\right] \\
&= \exp\left[-\frac{1}{2}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j^2+\vec{a}^{\mathrm{T}}\left(\Sigma_{\mathcal{F}}^{-1}+\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a}\right.\right. \\
&\qquad \left.\left. -\ (\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\vec{a}-\vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j)\right)\right] \qquad (40)
\end{aligned}$$

Completing the square in Equation 40 yields:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n)\propto\exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})-\hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}}+\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j^2\right)\right] \qquad (41)$$

where:

$$\Sigma_n^{-1}=\Sigma_{\mathcal{F}}^{-1}+\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}}) \qquad (42)$$

$$\hat{\vec{a}}=\Sigma_n\,(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j) \qquad (43)$$

Notice that as in the polynomial case (see Appendix A.1.1), neither of the two terms $\hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}}$ and $\frac{1}{\sigma_s^2}\sum_{j=1}^{n}y_j^2$ in Equation 41 depend on the RBF model parameters $\vec{a}$. This means we can simply remove the two "constant" terms from the exponent and introduce into Equation 41 the appropriate normalizing constants so $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ becomes a standard probability distribution on $\vec{a}$:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) = \frac{1}{(2\pi)^{K/2}|\Sigma_n|^{1/2}} \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \tag{44}$$

Thus, the RBF a-posteriori distribution is a multivariate Gaussian centered at $\hat{\vec{a}}$ (Equation 43) with covariance Σ_n (Equation 42).

A.2.2 The RBF EISD Measure

We now derive an expression for the *expected integrated squared difference* (EISD) between an unknown RBF target g and its *current* estimate given $\mathcal{D}_n$. We shall use this result later to derive $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$, the *total output uncertainty* cost function for RBF approximators.

The *expected integrated squared difference* (EISD) between an unknown target g and its estimate $\hat{g}$ given $\mathcal{D}_n$ is:

$$E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g},g)dg$$

where $\delta(\hat{g},g)$ is a standard integrated squared difference measure between two functions over the input space $\Re^d$ or some appropriate region of interest:

$$\delta(\hat{g},g) = \int_{\vec{x}\in\Re^d}(g(\vec{x})-\hat{g}(\vec{x}))^2 d\vec{x}.$$

For our RBF approximation function class, the *optimal* estimate for g given $\mathcal{D}_n$ has model parameters $\hat{\vec{a}}$ (Equation 43), since this is where the a-posteriori distribution $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ has a global maximum. Let $\hat{\vec{a}} = [\hat{a_0}\ \hat{a_1}\ \ldots\ \hat{a_K}]^{\mathrm{T}}$ and $\bar{\mathbf{z}} = [\mathcal{G}_1(\vec{x})\ \mathcal{G}_2(\vec{x})\ \ldots\ \mathcal{G}_K(\vec{x})]^{\mathrm{T}}$, one can rewrite $\delta(\hat{g},g)$ in terms of RBF model parameters as:

$$\begin{aligned}
\delta(\hat{\vec{a}},\vec{a}) &= \int_{\vec{x}\in\Re^d}[r(x,\vec{a})-r(x,\hat{\vec{a}})]^2\, d\vec{x} = \int_{\vec{x}\in\Re^d}\left[(\sum_{i=1}^{K} a_i\mathcal{G}_i(\vec{x})) - (\sum_{i=1}^{K}\hat{a}_i\mathcal{G}_i(\vec{x}))\right]^2\, d\vec{x} \\
&= \int_{\vec{x}\in\Re^d}\left[\sum_{i=1}^{K}(a_i-\hat{a}_i)\mathcal{G}_i(\vec{x})\right]^2\, d\vec{x} = \int_{\vec{x}\in\Re^d}(\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\bar{\mathbf{z}}\bar{\mathbf{z}}^{\mathrm{T}}(\vec{a}-\hat{\vec{a}})\, d\vec{x} \\
&= (\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\left(\int_{\vec{x}\in\Re^d}\bar{\mathbf{z}}\bar{\mathbf{z}}^{\mathrm{T}}\, d\vec{x}\right)(\vec{a}-\hat{\vec{a}}) \\
&= (\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}})
\end{aligned} \tag{45}$$

where $\mathbf{A}$ is a constant $K\times K$ matrix of numbers whose $(i,j)^{\mathrm{th}}$ element is:

$$\mathbf{A}(i,j) = \int_{\vec{x}\in\Re^d}\mathcal{G}_i(\vec{x})\mathcal{G}_j(\vec{x})\, d\vec{x}$$

Substituting Equations 44 and 45 into the EISD expression, we get:

$$
\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] = \int_{\vec{a}\in\Re^K} \mathcal{P}(\vec{a}|\mathcal{D}_n)\delta(\hat{\vec{a}},\vec{a})\, d\vec{a} \\
&= \int_{\vec{a}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\Sigma_n|^{1/2}} \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \\
&\qquad (\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}})\, d\vec{a}
\end{aligned} \tag{46}
$$

Making the following change of variables as in the polynomial case: $\vec{q} = \mathbf{A}^{\frac{1}{2}}(\vec{a}-\hat{\vec{a}})$, and noting that the integration bounds on $\vec{q}$ is still $\Re^K$, Equation 46 becomes:

$$
\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] \\
&= \int_{\vec{q}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\mathbf{A}|^{1/4}|\Sigma_n|^{1/2}|\mathbf{A}|^{1/4}} \\
&\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathrm{T}}\vec{q}\, d\vec{q} \\
&= \int_{\vec{q}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\Sigma_n\mathbf{A}|^{1/2}} \\
&\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathrm{T}}\vec{q}\, d\vec{q} \\
&= |\Sigma_n\mathbf{A}| \quad \propto \quad |\Sigma_n|
\end{aligned} \tag{47}
$$

since $\mathbf{A}$ is just a constant matrix of numbers.

Notice from Equation 42 that Σ_n depends only on the RBF function class priors $\Sigma_{\mathcal{F}}$, the K fixed Gaussian RBF kernels $\{\mathcal{G}_i(\cdot)|i=1,\ldots,K\}$, the output noise variance σ_s^2 and the previously sampled input locations $\{x_1, x_2, \ldots, x_n\}$. Like the polynomial case, it does not depend on the previous y data values actually observed. In other words, the previously observed y data values in $\mathcal{D}_n$ *do not* affect the EISD measure (Equation 47) for this Gaussian RBF concept class!

A.2.3 The RBF Total Output Uncertainty Measure

The *total output uncertainty* cost function is simply the *expected* EISD between g and its *new* estimate $\hat{g_{n+1}}$, if the learner samples next at $\vec{x_{n+1}}$. The cost function is given by Equation 7. We rewrite the expression below in terms of our RBF model parameters:

$$
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|\vec{x_{n+1}},\mathcal{D}_n)E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n\cup(\vec{x_{n+1}},y_{n+1})]\, dy_{n+1}. \tag{48}
$$

where:

$$
\mathcal{P}(y_{n+1}|\vec{x_{n+1}},\mathcal{D}_n) \propto \int_{\vec{a}\in\Re^K} \mathcal{P}(\mathcal{D}_n\cup(\vec{x_{n+1}},y_{n+1})|\vec{a})\mathcal{P}_{\mathcal{F}}(\vec{a})\, d\vec{a}.
$$

It is clear from Equation 48 that $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ is merely the *new* EISD weighted and averaged over all possible values of y_{n+1} at $\vec{x_{n+1}}$. Recall from Equation 42 however, that for this RBF concept class, the EISD between g and its estimate $\hat{g}$ depends only on the input $\vec{x_i}$

values in $\mathcal{D}_n$ and not on the observed y_i values. This means that $E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$, the new EISD resulting from sampling next at $\vec{x_{n+1}}$, does not depend on y_{n+1}! Equation 48 can therefore be further simplified, which leads to the following closed form expression for the *total output uncertainty* cost function, given also in Equation 17:

$$\begin{aligned}
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) &= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})] \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n)\, dy_{n+1} \\
&= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})] \\
&= |\Sigma_{n+1}\mathbf{A}| \quad \propto \quad |\Sigma_{n+1}| \qquad (49)
\end{aligned}$$

Here, Σ_{n+1} has exactly the same form as Σ_n in Equation 42, and depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the K fixed Gaussian RBF kernels $\{\mathcal{G}_i(\cdot)|i = 1, \ldots, K\}$, the output noise variance σ_s^2 and the data input locations
$\{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}, \vec{x_{n+1}}\}$.

References

[1] S. Ahmad and S. Omohundro. A Network for Extracting the Locations of Point Clusters using Selective Attention. Technical Report TR 90-011, International Computer Science Institute, University of California, Berkeley, 1990.

[2] D. Angluin. Learning k-term DNF Formulas using Queries and Counterexamples. Technical Report YALU/DCS/RR-559, Yale University, Department of Computer Science, 1987.

[3] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, April 1988.

[4] M. Bertero. Regularization Methods for Linear Inverse Problems. In C. Talenti, editor, *Inverse Problems*. Springer-Verlag, Berlin, 1986.

[5] D. Cohn. A Local Approach to Optimal Queries. In D. Touretzky, editor, *Proc. of 1990 Connectionist Summer School*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[6] V. Fedorov. *Theory of Optimal Experiments*, page 35. Academic Press, New York, 1972.

[7] Stuart Geman and Don Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[8] J. Hadamard. *La Theorie des Equations aux Derivees Partielles*. Editions Scientifique, Pekin, 1964.

[9] J. Hwang, J. Choi, S. Oh, and R. Marks. Query Learning based on Boundary Search and Gradient Computation of Trained Multi-layer Perceptrons. In *Proceedings IJCNN*, San Diego, CA, 1990. IEEE Press.

[10] D. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, Pasadena, CA, 1992.

[11] J. Marroquin, S. Mitter, and Tomaso Poggio. Probabilistic Solution of Ill-posed Problems in Computational Vision. In *Proceedings Image Understanding Workshop*, pages 293–309, Miami Beach, FL, December 1985.

[12] V. Morozov. *Methods of Solving Incorrectly posed Problems.* Springer-Verlag, Berlin, 1984.

[13] M. Plutowski and H. White. Active Selection of Training Examples for Network Learning in Noiseless Environments. Technical Report CS91-180, Department of Computer Science and Engineering, University of California, San Diego, 1991.

[14] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. Technical Report AIM–1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989.

[15] T. Poggio and F. Girosi. Extensions of a Theory of Networks for Approximation and Learning: Outliers and Negative Examples. Technical Report AIM–1220, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1990.

[16] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[17] C. Sammut and R. Banerji. Learning Concepts by Asking Questions. In J. Carbonell R. Michalski and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach (Vol. 2).* Morgan Kaufmann, Los Altos, CA, 1986.

[18] E. Shapiro. *Algorithmic Program Debugging.* MIT Press, Cambridge, MA, 1983.

[19] P. Sollich. Query Construction, Entropy, Generalization in Neural Network Models. *Physical Review E*, 49:4637–4651, 1994.

[20] K. Sung and T. Poggio. Example-based Learning for View-based Human Face Detection. In *Proceedings Image Understanding Workshop*, volume II, pages 843–850, Monterey, CA, November 1994.

[21] A. Tikhonov. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Math. Dokl.*, 4:1035–1038, 1963.

[22] A. Tikhonov and V. Arsenin. *Solutions of Ill-Posed Problems.* W. H. Winston, Washington, DC, 1977.

[23] L. Valiant. A Theory of Learnable. *Proc. of the 1984 STOC*, pages 436–445, 1984.

[24] L. Valiant. Learning Disjunctions of Conjunctions. In *Proceedings IJCAI*, pages 560–566, Los Angeles, CA, 1985.