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

A.I. Memo No. 1491 June 16, 1994
C.B.C.L. Paper No. 99

Neural Network Exploration Using Optimal Experiment Design

David A. Cohn
cohn@psyche.mit.edu

Abstract

We consider the question "How should one act when the only goal is to learn as much as possible?" Building on the theoretical results of Fedorov [1972] and MacKay [1992], we apply techniques from Optimal Experiment Design (OED) to guide the query/action selection of a neural network learner. We demonstrate that these techniques allow the learner to minimize its generalization error by exploring its domain efficiently and completely. We conclude that, while not a panacea, OED-based query/action has much to offer, especially in domains where its high computational costs can be tolerated.

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. The author was also funded by ATR Human Information Processing Laboratories, Siemens Corporate Research and NSF grant CDA-9309300.

1 Introduction

In many natural learning problems, the learner has the ability to act on its environment and gather data that will resolve its uncertainties. Most machine learning research, however, treats the learner as a passive recipient of data and ignores the role of this "active" component of learning. In this paper we employ techniques from the field of Optimal Experiment Design (OED) to guide the actions of a learner, selecting actions/queries that are statistically expected to minimize its uncertainty and error.

1.1 Active learning

Exploiting the active component of learning typically leads to improved generalization, usually at the cost of additional computation (see Figure 1) [Angluin, 1982; Cohn et al., 1990; Hwang et al., 1991].[1] There are two common situations where this tradeoff is desirable: In many situations the cost of taking an action outweighs the cost of the computation required to incorporate new information into the model. In these cases we wish to select queries judiciously, so that we can build a good model with the fewest data. This is the case if, for example, we are drilling oil wells or taking seismic measurements to locate buried waste. In other situations the data, although cheap, must be chosen carefully to ensure thorough exploration. Large amounts of data may be useless if they all come from an uninteresting part of the domain. This is the case with learning control of a robot arm: exploring by generating random motor torques can not be expected to give good coverage of the domain.

As computation becomes cheaper and faster, more problems fall within the realm where it is both desirable and practical to pursue active learning, expending more computation to ensure that one's exploration provides good data. The field of Optimal Experiment Design, which is concerned with the statistics of gathering new data, provides a principled way to guide this exploration. This paper builds on the theoretical results of Fedorov [1972] and MacKay [1992] to empirically demonstrate how OED may be applied to neural network learning, and to determine under what circumstances it is an effective approach.

The remainder of this section provides a formal problem definition, followed by a brief review of related work using optimal experiment design. Section 2 differentiates several classes of active learning problems for which OED is appropriate. Section 3 describes the theory behind optimal experiment design, and Section 4 demonstrates its application to the problems described in Section 2. Section 5 considers the computational costs of these experiments, and Section 6 concludes with a discussion of the results and implications for future work.

[1]In some cases active selection of training data can sharply reduce worst case computational complexity from NP-complete to polynomial time [Baum and Lang, 1991], and in special cases to linear time.

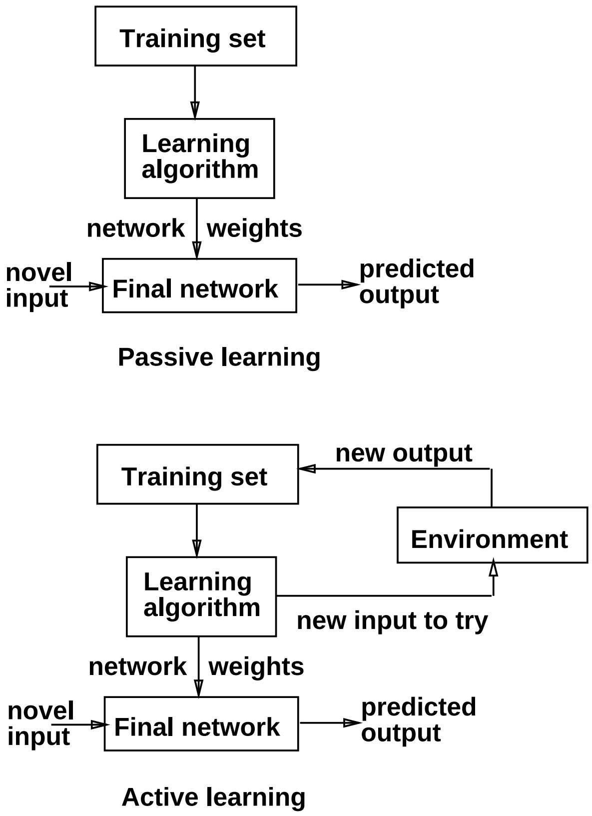


Figure 1: An active system will typically evaluate/train on its data iteratively, determining its next input based on the previous training examples. This iterative training may be computationally expensive, especially for learning systems like neural networks where good incremental algorithms are not available.

1.2 Problem definition

We consider the problem of learning an input-output mapping $X \rightarrow Y$ from a set of m training examples $\{(x_i, y_i)\}_{i=1}^m$, where $x_i \in X$, $y_i \in Y$.

We denote the parameterized learner $f_w()$, where $y = f_w(x)$ is the learner's output given input x and parameter vector w. The learner is trained by adjusting w to minimize the residual $S^2 = \frac{1}{2m}\sum_{i=1}^m (f_w(x_i) - y_i)^T (f_w(x_i) - y_i)$ on the training set. Let $\hat{w}$ be the weight vector that minimizes S^2. Then $\hat{y} = f_{\hat{w}}(x)$ is the learner's "best guess" of the mapping $X \rightarrow Y$: given x, $\hat{y}$ is an estimate of the corresponding y.

At each time step, the learner is allowed to select a new training input $\tilde{x}$ from a set of candidate inputs $\tilde{X}$. The selection of $\tilde{x}$ may be viewed as a "query" (as to an oracle), as an "experiment," or simply as an "action." Having selected $\tilde{x}$, the learner is given the corresponding $\tilde{y}$, and the resulting new example $(\tilde{x}, \tilde{y})$ is added to the training set. The learner incorporates the new data, selects another new $\tilde{x}$ and the process is repeated.

The goal is to choose examples that minimize the expectation of the learner's mean squared error $E_{MSE} = \left\langle (f_w(x) - y)^T (f_w(x) - y)\right\rangle_X$, where $\langle\cdot\rangle_X$ represents the expected value over X. In contrast to some other learning paradigms [Valiant, 1984; Blumer et al., 1986], we will assume that the input distribution $\mathcal{P}_X$ is known.[2] Below, we present several example problems:

Example 1: mapping buried waste. Consider a mobile sensor array traversing a landscape to map out subsurface electromagnetic anomalies. Its location at time t serves as input x_t, and the instrument reading at that location is output y_t. At the next time step, it can choose its new input $\tilde{x}$ from any location contiguous to its present position.

Example 2: robot arm dynamics. Consider learning the dynamics of a robot arm. The input is the state-action triplet $x_t = \{\Theta_t, \dot{\Theta}_t, \tau_t\}$, where Θ_t and $\dot{\Theta}_t$ are the arm's joint angles and velocities, respectively, and τ_t is the torque applied at time t. The output $y_t = \{\Theta_{t+1}, \dot{\Theta}_{t+1}\}$ is the resulting state. Note that here, although we may specify an arbitrary torque τ_t, the rest of the input, $\{\Theta_t, \dot{\Theta}_t\}$ is determined by y_{t-1}.

We emphasize that while the above problem definition has wide-ranging application, it is by no means all-encompassing. For some learning problems, we are not interested in the entire mapping $X \rightarrow Y$, but in finding the x that maximizes y. In this case, we may rely on the broad literature of optimization and response surface techniques [Box and Draper, 1969]. In other learning problems there may be additional constraints that must be considered, such as the need to avoid "failure" states. If the learner is required to perform as it learns (e.g. in a control task), we may also need to balance exploration and exploitation. Such constraints and costs may be incorporated into the data selection criterion as additional costs, but these issues are beyond the scope of this paper. In this paper we assume that the cost of the query $\tilde{x}$ is independent of $\tilde{x}$, and that the sole aim of active learning is to minimize E_{MSE}.

1.3 Related work with optimal experiment design

The literature on optimal experiment design is immense and dates back at least 50 years. We will just mention here a few closely related theoretical results and empirical studies; the interested reader should consult Atkinson and Donev [1992] for a survey of results and applications using optimal experiment design.

A canonical description of the theory of OED is given in Fedorov [1972]. MacKay [1992] showed that OED could be incorporated into a Bayesian framework for neural network data selection and described several interesting optimization criteria. Sollich [1994] considers the theoretical generalization performance of linear networks given greedy vs. globally optimal queries and varying assumptions on teacher distributions.

Empirically, optimal experiment design techniques have been successful when used for system identification tasks. In these cases a good parameterized model of the system is available, and learning involves finding the proper parameters. Armstrong [1989] used a form of OED to identify link masses and inertial moments of a robot arm, and found that automatically generated training trajectories provided a significant improvement over human-designed trajectories. Subrahmonia et al. [1992] successfully used experiment design to guide exploration of a sensor moving along the surface of an object parameterized as an unknown quadric.

Empirical work on using OED with neural networks is sparse. Plutowski and White [1993] successfully used it to filter an already-labeled data set for maximally informative points. Choueiki [1994] has successfully trained neural networks on quadratic surfaces with data drawn according to the D-optimality criterion, which is discussed in the appendix.

2 Learning with static and dynamic constraints

As seen in Section 1.2, different problems impose different constraints on the specification of $\tilde{x}$. These constraints may be classified as being either static or dynamic, and problems with dynamic constraints may be further divided according to whether or not the dynamics of the constraints are known a priori. The remainder of this section elaborates on these distinctions. Examples, with experimental results for each category, will be given in Section 4.

2.1 Active learning with static constraints

When a learner has static input constraints, its range of choices for $\tilde{x}$ is fixed, regardless of previous actions. Examples of problems with static constraints include setting mixtures of ingredients for an industrial process or selecting places to take seismic or electromagnetic measurements to locate buried waste.

The bulk of research on active learning per se has concentrated on learning with static constraints. In this setting, active learning algorithms are compared against algorithms learning from randomly chosen examples. In general, the number of randomly chosen examples needed to achieve an expected error of no more than ϵ scales as $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ [Blumer et al., 1989; Baum and Haussler 1989; Cohn and Tesauro, 1992; Haussler, 1992]. In some situations, active selection of training examples can reduce the sample complexity to $O(\log\frac{1}{\epsilon})$,[3] although worst case bounds for unconstrained querying are no better than those for choosing at random [Eisenburg and Rivest, 1990]. Average case analysis indicates that on many domains the expected performance of active selection of training examples *is* significantly better than that of random sampling [Freund and Seung, 1993]; these results have also been supported by empirical studies [Cohn et al., 1990; Hwang et al., 1991; Baum and Lang, 1991].

A limitation of the active learning algorithms mentioned above is that they are only applicable to specific active learning problems: the algorithms of Cohn et al., and Hwang et al. are limited to classification problems, and Baum and Lang's algorithm is further limited to a

[2] Both assumptions are reasonable in different situations; if we are attempting to learn to control a robot arm, for example, it is appropriate to assume that we know over what range we wish to control it.

[3] Consider cases where binary search is applicable.

specific network architecture (single hidden layer with sigmoidal units). The OED-based approach discussed in this paper is applicable to any network architecture whose output is differentiable with respect to its parameters, and may be used on both regression and classification problems.

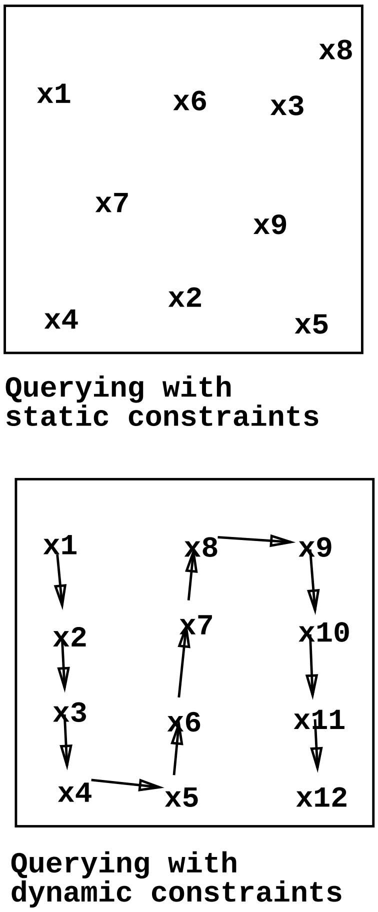


Figure 2: In problems with dynamic constraints, the set of candidate $\tilde{x}$ can change after each query. The $\tilde{x}_{t+1}$ accessible to the learner on the bottom depends on the choice made for $\tilde{x}_t$.

2.2 Active learning with dynamic constraints

In many learners, the constraints on $\tilde{x}$ are dynamic, and change over time. Inputs that are available to us on one time step may no longer be available on the next. Typically, these constraints represent some state of the system that is altered by the learner's actions. Training examples then describe a trajectory through state space.

In some such problems the dynamics of the constraints are known, and we may predict a priori what constraints we will face at time t, given an initial state and actions $x_1, x_2, \ldots, x_t$. Consider Example 1, using a mobile sensor array to locate buried waste. We can pre-plan the course the vehicle will take, but its successive measurements are constrained to lie in the neighborhood of previous ones. Alternatively, consider learning the forward kinematics of an arm: we specify joint angles Θ in an attempt to predict the arm's tip coordinates C. Barring any unknown obstacles, we can move from our current position Θ_t to a neighboring Θ_{t+1}, but can not select an arbitrary Θ_{t+1} for the next time step.

A more common, and more difficult problem is learning when the dynamics of the constraints are not known, and must be accommodated online. Learning the dynamics of a robot arm $\{\Theta_t, \dot{\Theta}_t, \tau_t\} \rightarrow \{\Theta_{t+1}, \dot{\Theta}_{t+1}\}$ is an example of this type of problem. At each time step t , the model input $\tilde{x}$ is a state-action pair $\{\Theta_t, \dot{\Theta}_t, \tau_t\}$, where Θ_t and $\dot{\Theta}_t$ are constrained to be the learner's current state. Until the action is selected and taken, the learner does not know what its new state, and thus its new constraints will be (this is in fact exactly what it is attempting to learn).

In most forms of constrained learning problems, random exploration is a poor strategy. Taking random actions leads to a form of "drunkard's walk" over X, which can require an unacceptably large number of examples to give good coverage [Whitehead, 1991].

In cases where the dynamics of the constraints are known a priori, we can plan a trajectory that will uniformly cover X in some prespecified number of steps. In general, though, we will have to resort to some online process to decide "what to try next." Some successful heuristic exploration strategies include trying to visit unvisited states [Schaal and Atkeson, 1994], trying to visit places where we perform poorly [Linden and Weber, 1993], taking actions that improved our performance in similar situations [Schmidhuber and Storck, 1993], or maintaining a heuristic "confidence map" [Thrun and Möller, 1992]. Some researchers, in cases where the exploration is considered a secondary problem, provide the learner with a uniformly distributed training set, in effect assuming the problem allows unconstrained querying (e.g. Mel [1992]).

An important limitation of the above work with dynamic constraints is that, for the most part, the methods are restricted to discrete state spaces. Continuous state and action spaces must be accommodated either through arbitrary discretization or through some form of on-line partitioning strategy, such as Moore's Parti-Game algorithm [Moore, 1994]. The OED-based approach discussed in this paper is, by nature, applicable to domains with both continuous state and action spaces.

3 Data selection according to OED

In this section, we review the theory of optimal experiment design applied to neural network learning. As stated in the introduction, our primary goal is to minimize E_{MSE}. An alternative goal of system identification is discussed briefly in the appendix, and other interesting goals, such as eigenvalue maximization and entropy minimization, may be found in Fedorov [1972] and MacKay [1992].

Error minimization is pursued in the OED framework by selecting data to minimize model uncertainty. Uncertainty in this case is manifested as the learner's estimated output variance $\sigma^2_{\hat{y}}$. The justification for selecting data to minimize variance comes from the nature of MSE.

Defining $\overline{y}|x = \langle \hat{y}|x \rangle_Y$, mean squared error may be decoupled into variance and bias terms.

$$\begin{aligned} E_{MSE} &= \left\langle (\hat{y}|x - y|x)^2 \right\rangle_X \\ &= \left\langle (\hat{y}|x - \overline{y}|x)^2 \right\rangle_X + \left\langle (\overline{y}|x - y|x)^2 \right\rangle_X \\ &= \sigma_{\hat{y}}^2 + \left\langle (\overline{y}|x - y|x)^2 \right\rangle_X . \end{aligned}$$

Given an unbiased estimator, or an estimator for which either the bias is small or independent of the training set, error minimization amounts to minimizing the variance of the estimator. For the rest of our computations we will neglect the bias term, and select data solely to minimize the estimated variance of our learner.[4] An in-depth discussion of bias/variance tradeoff may be found in Geman et al. [1992].

3.1 Estimating variance

Estimates for $\sigma_{\hat{y}}^2$ may be obtained by adopting techniques derived for linear systems. We write the network's output sensitivity as $g(x) = \partial \hat{y}|x / \partial w = \partial f_{\hat{w}}(x)/\partial w$, and define the Fisher Information Matrix to be

$$\begin{aligned} A &= \frac{1}{S^2}\frac{\partial^2 S^2}{\partial w^2} \\ &= \frac{1}{S^2}\sum_{i=1}^{m}\left[\frac{\partial \hat{y}|x_i}{\partial w}\frac{\partial \hat{y}|x_i}{\partial w}^T + (\hat{y}|x_i - y_i)\frac{\partial^2 \hat{y}|x_i}{\partial w^2}\right] \\ &\approx \frac{1}{S^2}\sum_{i=1}^{m} g(x_i)g(x_i)^T . \qquad (1) \end{aligned}$$

The approximation in Equation 1 holds when the network fits the data well or the error surface has relatively constant curvature in the vicinity of $\hat{w}$. We may then write the parameter covariance matrix as $\sigma_{\hat{w}}^2 = A^{-1}$ and the output variance at reference input x_r as

$$\sigma_{\hat{y}|x_r}^2 \approx g(x_r)^T A^{-1} g(x_r) \qquad (2)$$

subject to the same approximations (see Thisted [1988] for derivations).[5] Note that the estimate $\sigma_{\hat{y}|x_r}^2$ applies only to the variance at a particular reference point. Our interest is in estimating $\sigma_{\hat{y}}^2$, the average variance over all of X. We do not have a method for directly integrating over X, and instead opt for a stochastic estimate based on an average of $\sigma_{\hat{y}|x_r}^2$, with x_r drawn according to $\mathcal{P}_X$. Writing the first and second moments of g as $\overline{g} = \langle g(x) \rangle_X$ and $\overline{gg^T} = \left\langle g(x)g(x)^T \right\rangle_X$, this estimate can be computed efficiently as

$$\left\langle \sigma_{\hat{y}}^2 \right\rangle_X = \overline{g}^T A^{-1}\overline{g} + tr(A^{-1}\overline{gg^T}), \qquad (3)$$

where $tr()$ is the matrix trace. Instead of recomputing Equation 2 for each reference point, $\overline{g}$ and $\overline{gg^T}$ may be computed over the reference points, and Equation 3 evaluated once.

[4] The bias term will in fact reappear as a limiting factor in the experimental results described in Section 4.2.

[5] If the inverse A^{-1} does not exist, then the parameter covariances are not well-defined. In practice, one could use the pseudo-inverse, but the need to this arose very rarely in our experiments, even with small training sets.

3.2 Quantifying change in variance

When an input $\tilde{x}$ is queried, we obtain the resulting output $\tilde{y}$. When the new example $(\tilde{x}, \tilde{y})$ is added to the training set, the variance of the model will change. We wish to select $\tilde{x}$ optimally, such that the resulting variance, denoted $\tilde{\sigma}_{\hat{y}}^2$, is minimized.

The network provides a (possibly inaccurate) estimate of the distribution $\mathcal{P}(\tilde{y}|\tilde{x})$, embodied in an estimate of the mean $(\hat{y}|\tilde{x})$ and variance (S^2). Given infinite computational power then, we could use these estimates to stochastically approximate $\tilde{\sigma}_{\hat{y}|x}^2$ by drawing examples according to our estimate of $\mathcal{P}(\tilde{y}|\tilde{x})$, training on them, and averaging the new variances. In practice though, we must settle for a coarser approximation. Note that the approximation in Equation 1 is independent of the actual y_i values of the training set; the dependence is implicit in the choice of $\hat{w}$ that minimizes S^2. If $\mathcal{P}(\tilde{y}|\tilde{x})$ conforms to our expectations, $\hat{w}$ and $g()$ will remain essentially unchanged, allowing us to compute the new information matrix $\tilde{A}$ as

$$\tilde{A} \approx A + \frac{1}{S^2} g(\tilde{x})g(\tilde{x})^T . \qquad (4)$$

From the new information matrix, we may compute the new parameter variances, and from there, the new output variances. By the matrix inversion lemma

$$\begin{aligned} \tilde{A}^{-1} &= \left(A + \frac{1}{S^2}\, g(\tilde{x})g(\tilde{x})^T\right)^{-1} \\ &= A^{-1} - \frac{A^{-1}g(\tilde{x})g(\tilde{x})^T A^{-1}}{S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x})} . \qquad (5) \end{aligned}$$

The utility of querying at $\tilde{x}$ may be expressed in terms of the expected change in the estimated output variance $\sigma_{\hat{y}}^2$. The expected new output variance at reference point x_r is

$$\begin{aligned} \left\langle \tilde{\sigma}_{\hat{y}|x_r}^2 \right\rangle_{\tilde{Y}} &= g(x_r)^T \tilde{A}^{-1} g(x_r) \\ &= g(x_r)^T \left[A^{-1} - \frac{A^{-1}g(\tilde{x})g(\tilde{x})^T A^{-1}}{S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x})}\right] g(x_r) \\ &= g(x_r)^T A^{-1} g(x_r) - \frac{\left[g(x_r)^T A^{-1} g(\tilde{x})\right]^2}{S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x})} \\ &= \sigma_{\hat{y}|x_r}^2 - \frac{\sigma_{\hat{y}|x_r\tilde{x}}^2}{S^2 + \sigma_{\hat{y}|\tilde{x}}^2}, \end{aligned}$$

where $\sigma_{\hat{y}|x_r\tilde{x}}$ is defined as $g(x_r)^T A^{-1} g(\tilde{x})$. Thus, when $\tilde{x}$ is queried, the expected change in output variance at x_r is

$$\left\langle \Delta\sigma_{\hat{y}|x_r}^2 \right\rangle_{\tilde{Y}} |\tilde{x} = \frac{\sigma_{\hat{y}|x_r\tilde{x}}^2}{S^2 + \sigma_{\hat{y}|\tilde{x}}^2}, \qquad (6)$$

We compute $\left\langle \Delta\sigma_{\hat{y}}^2 \right\rangle |\tilde{x}$ as a stochastic approximation from $\left\langle \Delta\sigma_{\hat{y}|x_r}^2 \right\rangle |\tilde{x}$ for x_r drawn from $\mathcal{P}_X$. Reusing the estimate $\overline{gg^T}$ from the previous section, we can write the expectation of Equation 6 over X as

$$\left\langle \Delta\sigma_{\hat{y}}^2 |\tilde{x} \right\rangle_{X,\tilde{Y}} = \frac{g(\tilde{x})^T A^{-1}\overline{gg^T} A^{-1} g(\tilde{x})}{S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x})} . \qquad (7)$$

3.3 Selecting an optimal $\tilde{\mathbf{x}}$

Given Equation 7, the problem remains of how to select an $\tilde{x}$ that maximizes it. One approach to selecting a next input is to use selective sampling: evaluate a number of possible random $\tilde{x}$, then choose the one with the highest expected gain. This is efficient so long as the dimension of the action space is small. For high-dimensional problems, we may use gradient ascent to efficiently find good $\tilde{x}$. Differentiating Equation 7 with respect to $\tilde{x}$ gives a gradient

$$\nabla_{\tilde{x}} \left\langle \Delta\sigma_{\hat{y}}^2 \right\rangle_{X,\tilde{Y}} = \frac{2g(\tilde{x})^T A^{-1} \overline{gg^T} A^{-1}}{(S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x}))^2} \frac{\partial g(\tilde{x})}{\partial \tilde{x}}. \tag{8}$$

We can "hillclimb" on this gradient to find a $\tilde{x}$ with a locally optimal expected change in average output variance.

It is worth noting that both of these approaches are applicable in continuous domains, and therefore well-suited to problems with continuous action spaces. Furthermore, the gradient approach is effectively immune to the overabundance of candidate actions in high-dimensional action spaces.

3.4 A caveat: *greedy* optimality

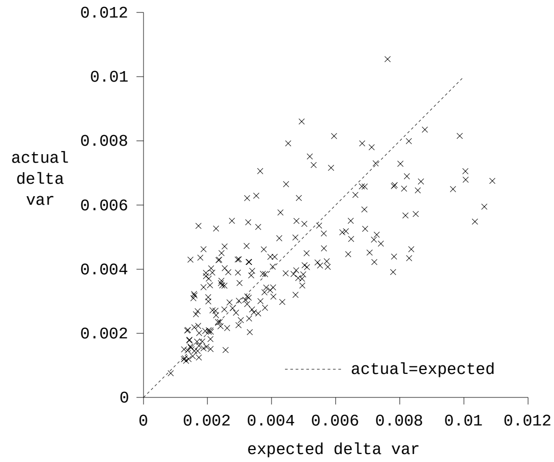
We have described a criterion for one-step, or *greedy* optimization. That is, each action/query is chosen to maximize the change in variance on the next step, without regard to how future queries will be chosen. The globally optimal, but computationally expensive approach would involve optimizing over an entire trajectory of m actions/queries. Trajectory optimization entails starting with an initial trajectory, computing the expected gain over it, and iteratively relaxing points on the trajectory towards optimal expected gains (subject to other points along the trajectory being explored). After the iteration has settled, the first point in the trajectory is queried, and the relaxation is repeated on the remaining part of the trajectory. Experiments using this form of optimization did not demonstrate measurable improvement, in the average case, over the greedy method, so it appears that trajectory optimization may not be worth the additional computational expense, except in extreme situations (see Sollich [1994] for a theoretical comparison of greedy and globally-optimal querying).

4 Experimental Results

In this section, we describe two sets of experiments using optimal experiment design for error minimization. The first attempts to confirm that the gains predicted by optimal experiment design may actually be realized in practice, and the second applies OED to learning tasks with static and dynamic constraints. All experiments described in this section were run using feedforward networks with a single hidden layer of 20 units. Hidden and output units used the 0-1 sigmoid as a nonlinearity. All runs were performed on the Xerion simulator [van Camp et al., 1993] using the default weight update rule ("Rudi's Conjugate Gradient" with "Ray's Line Search") with no weight decay term.

4.1 Expected versus actual gain

It must be emphasized that the gains predicted by OED are *expected* gains. These expectations are based on the series of approximations detailed in the previous section, which may compromise the realization of any actual gain. In order for the expected gains to materialize, two "bridges" must be crossed. First, the expected decrease in model variance must be realized as an actual decrease in variance. Second, the actual decrease in model variance must translate into an actual decrease in model MSE.



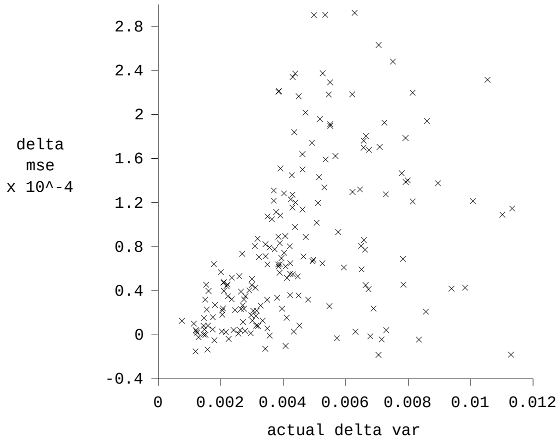


Figure 3: (top) Correlations between expected change in output variance and actual change output variance. (bottom) Correlations between actual change in output variance and change in mean squared error. Correlations are plotted for a network with a single hidden layer of 20 units trained on 50 examples from the arm kinematics task.

4.1.1 Expected decreases in variance → actual decreases in variance

The translation from expected to actual changes in variance requires coordination between the exploration strategy and the learning algorithm: to predict how the variance of a weight will change with a new piece of data, the predictor must know how the weight itself (and its neighboring weights) will change. Using a black box routine like backpropagation to update the weights virtually guarantees that there will be some mismatch between expected and actual decreases in variance. Experiments indicate that, in spite of this, the correlation between predicted and actual changes in variance are relatively good (Figure 3a).

4.1.2 Decreases in variance → decreases in MSE

A more troubling translation is the one from model variance to model correctness. Given the highly nonlinear nature of a neural network, local minima may leave us in situations where the model is very confident but entirely wrong. Due to high confidence, the learner may reject actions that would reduce its mean squared error and explore areas where the model is correct, but has low confidence. Evidence of this behavior is seen in the lower right corner of Figure 3b, where some actions which produce a large decrease in variance have little effect on E_{MSE}. This behavior appear to be a manifestation of the bias term discussed in Section 3; these queries reduce variance while increasing the learner's bias, with no net decrease in error. While this demonstrates a weak point in the OED approach (which will be further illustrated below), we find in the remainder of this section that its effect is negligible for many classes of problems.

4.2 Querying with static constraints

Here we consider a simple learning problems with static constraints: learning the forward kinematics of a planar arm from examples. The input $X = \{\Theta_1, \Theta_2\}$ specified the arm's joint angles, and the learner attempted to learn a map from these to the Cartesian coordinates $Y = \{C_1, C_2\}$ of the arm's tip. The "shoulder" and "elbow" joints were constrained to the $0-360^o$ and $0-180^o$ respectively; on each time step the learner was allowed to specify an arbitrary $\tilde{x} \in X$ within those limits.

For the greedy OED learner, $\tilde{x}$ was chosen by beginning at a random point in X and hillclimbing the gradient of Equation 6 to a local maximum before querying. This strategy was compared with simply choosing $\tilde{x}$ at random, and choosing $\tilde{x}$ according to a uniform grid over X.[6]

We compared the variance and MSE of the OED-based learner with that of the random and grid learners. The average variance of the OED-based learner was almost identical to that of the grid learner and slightly better than that of the random learner (Figure 4b). In terms of MSE however, the greedy OED learner did not fare as well. Its error was initially comparable to that of the grid strategy, but flattened out at an error approximately twice that of the asymptotic limit (Figure 4b). This flattening appears to be a result of bias. As discussed in Section 3, the network's error is composed of a variance term and a bias term, and the OED-based approach, while minimizing variance, appears (in this case) to leave a significant amount of bias.

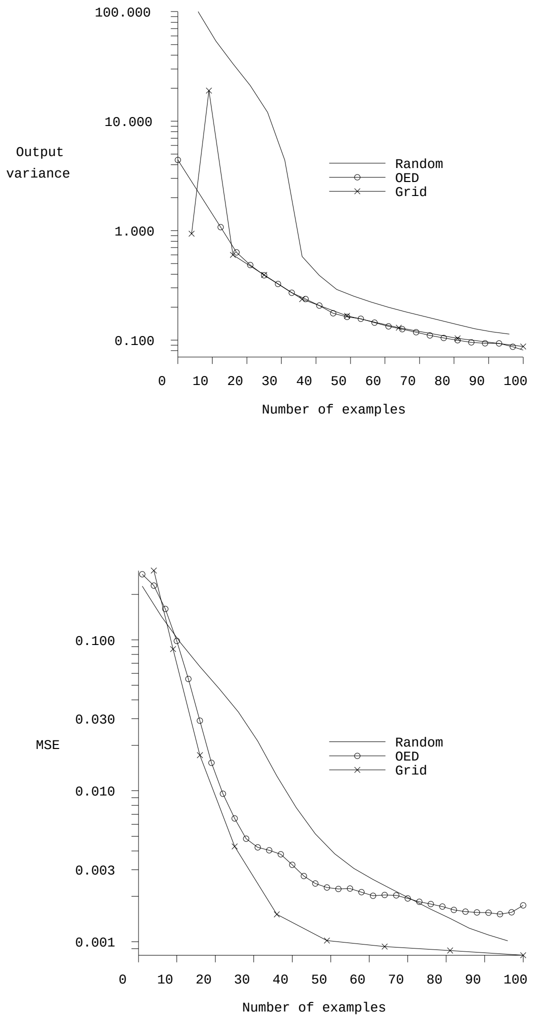


Figure 4: Querying with static constraints to learn the kinematics of a planar two-joint arm. (top) Variance using OED-based actions is better than that using random queries, and matches the variance of a uniform grid. (bottom) MSE using OED-based actions is initially very good, but breaks down at larger training set sizes. Curves are averages over six runs apiece for OED and grid learners, and 12 runs for the random learner.

[6] Note the uniform grid strategy is not viable for incrementally drawn training sets – the size of the grid must be fixed before any examples are drawn. In these experiments, entirely new training sets of the appropriate size were drawn for each new grid.

4.3 Querying with dynamic constraints

For learning with dynamic constraints, we again used the planar arm problem, but this time with a more realistic restriction on new inputs. For the first series of experiments, the learner learned the kinematics by incrementally adjusting Θ_1 and Θ_2 from their values on the previous query. The limits of allowable movement on each step corresponded to constraints with known dynamics. The second set of experiments involved learning the dynamics of the same arm based on torque commands. The unknown next state of the arm corresponded to constraints with unknown dynamics.

4.3.1 Constraints with known dynamics

To learn the arm kinematics, the learner hillclimbed to find the Θ_1 and Θ_2 within its limits of movement that would maximize the stochastic approximation of Δvar. On each time step Θ_1 and Θ_2 were limited to change by no more than $\pm 36^o$ and $\pm 18^o$ respectively.

We compared variance and MSE of the OED-based learner with that of an identical learner which explored randomly by "flailing," and with a learner trained on a series of hand-tuned trajectories.

The greedy OED-based learner found exploration trajectories that, intuitively, appear to give good global coverage of the domain (see Figure 5). In terms of performance, the average variance over the OED-based trajectories was almost as good as that of the best hand-tuned trajectory, and both were far better than that of the random exploration trajectories. In terms of MSE, the average error over OED-based trajectories was almost as good as that of the best hand-tuned trajectory, and again, both were far better than the random exploration trajectories (Figure 6). Note that in this case, bias does not seem to play a significant role. We discuss the performance and computational complexity of this task in greater detail in Section 5.

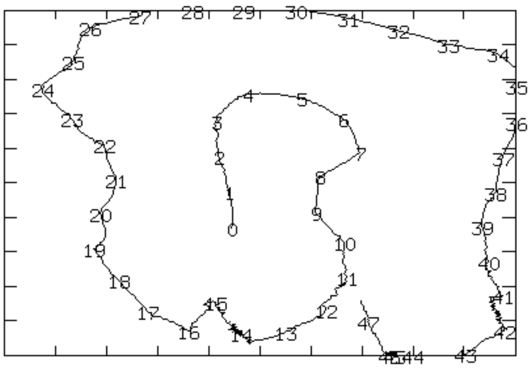


Figure 5: Querying with dynamic constraints: learning 2D arm kinematics. Example of OED-based learner's trajectory through angle-space.

4.3.2 Constraints with unknown dynamics

For this set of experiments, we once again used the planar two-jointed arm, but now attempted to learn the arm dynamics. The learner's input $X = \{\Theta_1, \Theta_2, \dot{\Theta}_1, \dot{\Theta}_2, \tau_1, \tau_2\}$ specified the joint positions, velocities and torques. Based on these, the learner attempted to learn the arm's next state $Y = \{\Theta_1', \Theta_2', \dot{\Theta}_1', \dot{\Theta}_2'\}$. As with the kinematics experiment, we compared random exploration with the greedy OED strategy described in the previous section. Without knowing the dynamics of the input constraints, however, we do not have the ability to specify a preset trajectory.

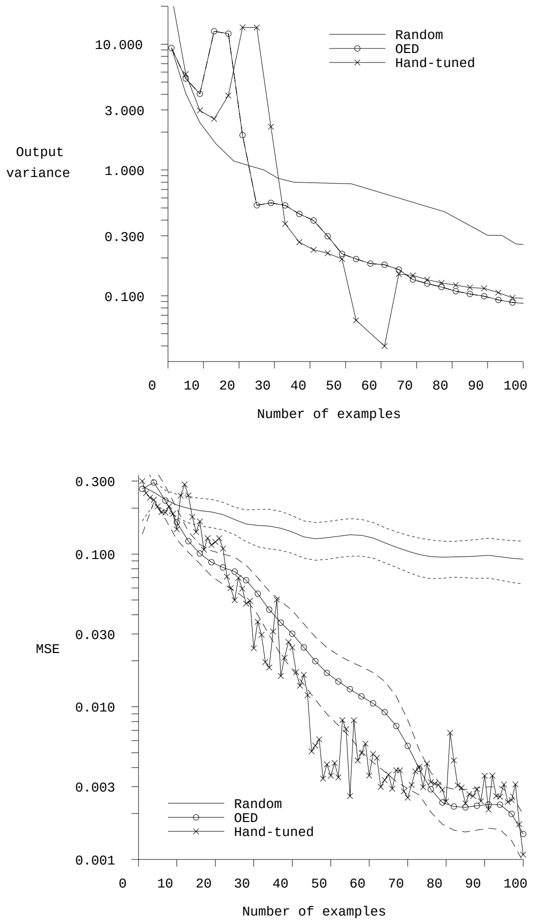


Figure 6: Querying with dynamic constraints: learning 2D arm kinematics. (top) Variance using greedy OED actions is better than that using random exploration, and matches the variance of the best hand-tuned trajectory. (bottom) MSE using greedy OED-based exploration is much better than that of random exploration and almost as good as that of the best hand-tuned trajectory. Curves are averages over 5 runs apiece for OED-based and random exploration.

The performance of the learner whose exploration was guided by OED was asymptotically much better than that of the learner following a random search strategy (Figure 7). It is instructive to notice, however, that this improvement is not immediate, but appears only after the learner has taken a number of steps.[7] Intuitively,

[7] This behavior is visible in the other problem domains as

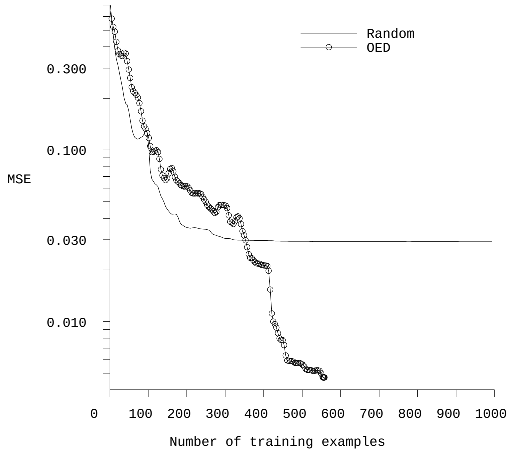


Figure 7: MSE of forward dynamic model for two-joint planar arm.

this may be explainable by the assumptions made in the OED formalism: the network uses its estimate of variance of the current model to determine what data will minimize the variance. Until there is enough data for the model to become reasonably accurate, the estimates will be correspondingly inaccurate, and the search for "optimal" data will be misled. It would be useful to have a way of determining at what point the learner's estimates become reliable, so that one could explore randomly at first, then switch to OED-guided exploration when the learner's model is accurate enough to take advantage of it.

5 Computational costs and approximations

The major concern with applying the OED techniques described in this paper is computational cost. In this section we consider the computational complexity of selecting actions via OED techniques, and consider several approximations aimed at reducing the computational costs. These costs are summarized in Table 1, with the time constants observed for runs performed on a Sparc 10.

We divide the learning process in three steps: training, variance estimation, and data selection. We show that, for the case examined, in spite of increased complexity, the improvement in performance more than warrants the use of OED for data selection.

Cost of training: Two training regimens were tested for the OED-guided learners: batch training reinitialized after each new example was added, and incremental training, reusing the previous network's weights after each new example. While the batch-trained learners' performance was slightly better, their total training time was significantly longer than their incrementally trained counterparts (Figure 8).

well, but is not as pronounced.

operation	constant	order
Batch train	0.029	mn
Incremental train	0.093	n
Compute exact A	$3x10^{-7}$	mn^3
Compute approx. A	$7.2x10^{-6}$	mn^2
Invert to get A^{-1}	$3.2x10^{-7}$	n^3
Compute $var(x_r)$	$5.0x10^{-6}$	n^2
Compute $E[\Delta var(X)\|x]$	$5.4x10^{-6}$	rn^2
Compute gradient	$1.9x10^{-5}$	rn^2

Table 1: Typical compute times, in seconds, for operations involved in selecting new data and training. Number of weights in network = n, number of training examples = m, and number of reference points (at which variance or gradient is measures) = r. Time constants are for runs performed on a Sparc 10 using the Xerion simulator.

Cost of variance estimation: (Equation 3) Variance estimation requires computing and inverting the Hessian. The inverse Hessian may then be used for an arbitrary number of variance estimates and must only be recomputed when the network weights are updated. The approximate Hessian of Equation 1 may be computed in time $O(mn^2)$, but the major cost remains the inversion. We have experimented with diagonal and block diagonal Hessians, which may be inverted quickly, but without the off-diagonal terms, the learner failed to generate reasonable training sets. Recent work by Pearlmutter [1994] offers a way to bring the cost of computing the first term of Equation 3, but computing the second term remains an $O(n^3)$ operation.

Cost of data selection: (Equations 6, 7 and 8) Computing Equation 6 is an $O(n^2)$ operation, which must be performed on each of r reference points, and must be repeated for each candidate $\tilde{x}$. Alternatively, the "moment-based" selection (Equation 7) and gradient methods (Equation8) both require an $O(n^3)$ matrix multiplication which must be done once, after which any number of iterations may be performed with new $\tilde{x}$ in time $O(n^2)$. Using Perlmutter's approach to directly approximate $A^{-1}g(\tilde{x})$ would allow an approximation of Equation 7 to be computed in $O(n^2)$ times an "accuracy" constant. We have not yet determined what effect this time/accuracy tradeoff has on network performance.

The payoff: cost vs. performance. Obviously, the OED-based approach requires significantly more computation time than does learning from random examples. The payoff comes when relative performance is considered. We turn again to the kinematics problem discussed in Section 4.3.1. The approximate total time involved in training a learner on 100 random training examples from this problem (as computed from Table 1) is 170 seconds. For "full-blown" OED, using incremental training, the total time is 790 seconds. As shown in Figure 8, exploring randomly causes our MSE to decrease roughly as an inverse polynomial, while the various OED strategies decrease MSE roughly exponentially in the number of examples. To achieve the MSE reached by training on

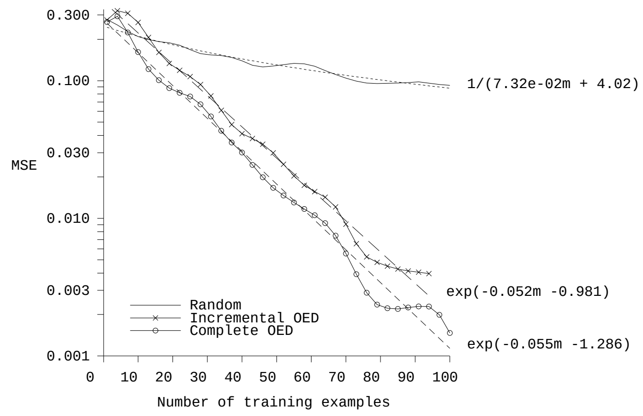


Figure 8: Learning curves for the kinematics problem from Section 4.2. Best fit functional forms are plotted for random exploration, incrementally-trained OED and OED completely retrained on new data set.

OED-selected data, we would need to train on approximately 3380 randomly selected data examples. This would take approximately 7500 seconds, over two hours! With this much data, the training time alone is greater than the total OED costs, so regardless of data costs, selecting data via OED is the preferable approach.

With the kinematics example there is the option of hand-tuning a learning trajectory, which requires no more data than the OED approach, and can nominally be learned in less time. This, however, required hours of human intervention to repeatedly re-run the simulations trying different preset exploration trajectories. In the dynamics example and in other cases where the state transitions are unknown, preset exploration strategies are not an option; we must rely on an algorithm for deciding our next action, and the OED-based strategy appears to be a viable, statistically well-founded choice.

6 Conclusions and Future Work

The experiments described in this paper indicate that, for some tasks, optimal experiment design is a promising tool for guiding active learning in neural networks. It requires no arbitrary discretization of state or action spaces, and is amenable to gradient search techniques. The appropriateness of OED for exploration hinges on the two issues described in the previous two sections: the nature of the input constraints and the computational load one is able to bear.

For learning problems with static constraints, the advantage of applying OED, or any form of intelligent active learning appears to be problem dependent. Random exploration appears to be reasonably good at decreasing variance, and as seen in Section 4.2, appears to decrease bias as well. For a problem where learner bias is likely to be a major factor, the advantages of the OED approach are unclear.

The real advantage of the OED-based approach appears to lie in problems where the input constraints are dynamic, and where random actions fail to provide good exploration. Compared with arbitrary heuristics, the OED-based approach has the arguable advantage of being the "right thing to do," in spite of its computational costs.

The cost, however, is a major drawback. A decision time on the order of 1-10 seconds may be sufficient for many applications, but is much too long to guide real-time exploration of dynamical systems such as robotic arms. The operations required for hessian computation and data selection may be efficiently parallelized; the remaining computational expense lies in retraining the network to incorporate each new example. The retraining cost, which is common to all on-line neural exploration algorithms, may be amortized by selecting queries/actions in small batches rather than purely sequentially. This "semi-batched" approach is a promising direction for future work.

Another promising direction, which offers hope of even greater speedups than the semi-batch approach, is switching to an alternative, entirely non-neural learner with which to pursue exploration.

6.1 Improving performance with alternative learners

We may be able to bring down computational costs *and* improve performance by using a different architecture for the learner. With a standard feedforward neural network, not only is the repeated computation of variances expensive, it sometimes fails to yield estimates suitable for use as confidence intervals (as we saw in Section 4.1.2). A solution to both of these problems may lie in selection of a more amenable architecture and learning algorithm. Two such architectures, in which output variances have a direct role in estimation, are mixtures of Gaussians [McLachlan and Basford, 1988; Nowlan, 1991; Ghahramani and Jordan, 1994] and locally weighted regression [Cleveland et al., 1988; Schaal and Atkeson, 1994]. Both have excellent statistical modeling properties, and are computationally more tractable than feedforward neural networks. We are currently pursuing the application of optimal experiment design techniques to these models and have observed encouraging preliminary results [Cohn et al., 1994].

6.2 Active elimination of bias

Regardless of which learning architecture is used, the results in Section 4.2 make it clear that minimizing variance alone is not enough. For large, data-poor problems, variance will likely be the major source of error, but as variance is removed (via the techniques described in this paper), the bias will constitute a larger and larger portion of the remaining error.

Bias is not as easily estimated as variance; it is usually estimated by expensive cross validation, or by running ensembles of learners in parallel (see, e.g. Geman et al. [1992] and Connor [1993]). Future work will need to include methods for efficiently estimating learner bias and taking steps to ensure that it too is minimized in an optimal manner.

Acknowledgements

I am indebted to Michael I. Jordan and David J.C. MacKay for their help in making this research possible. Thanks are also due to the University of Toronto and the Xerion group for use of and assistance with the Xerion simulator, and to Cesare Alippi for useful comments on an earlier draft of this paper.

References

D. Angluin. (1982) A note on the number of queries needed to identify regular languages. *Inform. Control*, **51**:76–87.

B. Armstrong. (1989) On finding exciting trajectories for identification experiments. *Int. J. of Robotics Research*, **8**(6):28–48.

A. Atkinson and Donev. (1992) Optimum experimental designs, Clarendon Press, New York.

J. Connor. (1993) Bootstrap methods in neural network time series prediction. In J. Alspector et al., eds., *Proceedings of the International Workshop on Application of Neural Networks to Telecommunications*, Lawrence Erlbaum, Hillsdale, NJ.

J. Craig. (1989) Introduction to robotics, Addison-Wesley, New York.

E. Baum and D. Haussler. (1989) What size net gives valid generalization? In D. Touretzky, ed., *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann, San Francisco, CA.

E. Baum and K. Lang. (1991) Constructing hidden units using examples and queries. In R. Lippmann et al., eds., *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann, San Francisco, CA.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. (1989) Learnability and the Vapnik-Chervonenkis dimension. *JACM* **36**(4):929–965.

G. Box and N. Draper. (1969) *Evolutionary operation.* Wiley, New York.

H. Choueiki. (1994) Doctoral dissertation, in preparation. Department of Industrial and Systems Engineering, Ohio State University.

W. Cleveland, S. Devlin and E. Grosse. (1988) Regression by local fitting. *Journal of Econometrics*, **37**:87–114.

D. Cohn, L. Atlas and R. Ladner. (1990) Training connectionist networks with queries and selective sampling. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Francisco, CA.

D. Cohn, Z. Ghahramani and M. Jordan. (1994) Active learning with statistical models. Submitted to Advances in Neural Information Processing Systems 7, Morgan Kaufmann, San Francisco, CA.

B. Eisenberg and R. Rivest. (1990) On the sample complexity of pac-learning using random and chosen examples. In M. Fulk and J. Case, eds., *ACM 3rd Annual Workshop on Computational Learning Theory*, Morgan Kaufmann, San Francisco, CA.

V. Fedorov. (1972) *Theory of Optimal Experiments.* Academic Press, New York.

Y. Freund, H. S. Seung, E. Shamir and N. Tishby. (1993) Information, prediction, and query by committee. In S. Hanson et al., eds., *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Francisco, CA.

S. Geman, E. Bienenstock and R. Doursat. (1992) Neural networks and the bias/variance dilemma. *Neural Computation*, **4**(1):1–58.

Z. Ghahramani and M. Jordan. (1994) Supervised learning from incomplete data via an EM approach. In J. Cowan et al., eds., *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Francisco, CA.

D. Haussler. (1992) Generalizing the pac model for neural nets and other learning applications. *Information and Computation*, **100**(1):78–150.

R. Jacobs, M. Jordan, S. Nowlan and G. Hinton. (1991) Adaptive mixtures of local experts. *Neural Computation* **3**:79–87.

M. Jordan and D. Rumelhart. (1992) Forward models: Supervised learning with a distal teacher. *Cognitive Science*, **16**(3):307–354.

M. Kuperstein. (1988) Neural model of adaptive hand-eye coordination for single postures. *Science*, **239**:1308–1311.

A. Linden and F. Weber. (1993) Implementing Inner Drive by Competence Reflection, in H. Roitblat et al., eds., *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*, MIT Press, Cambridge, MA.

D. MacKay. (1992) Information-based objective functions for active data selection, *Neural Computation* **4**(4): 590–604.

G. McLachlan and K. Basford. (1988) *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker.

B. Mel. (1992) Connectionist robot motion planning: a neurally-inspired approach to visually-guided reaching. Academic Press, Boston, MA.

A. Moore. (1994) The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In J. Cowan et al., editors, *Advances in Neural Information Processing Systems 6.* Morgan Kaufmann, San Francisco, CA.

S. Nowlan. (1991) Soft Competitive Adaptation: Neural Network Learning Algorithms based on Fitting Statistical Mixtures. CMU-CS-91-126, School of Computer

Science, Carnegie Mellon University, Pittsburgh, PA.

B. Pearlmutter. (1994) Fast Exact Multiplication by the Hessian. *Neural Computation*, **6**(1):147–160.

M. Plutowski and H. White. (1993) Selecting concise training sets from clean data. *IEEE Trans. on Neural Networks*, **4**(2):305–318.

S. Schaal and C. Atkeson. (1994) Robot Juggling: An Implementation of Memory-based Learning. *Control Systems* **14**(1):57–71.

J. Schmidhuber and J. Storck. (1993) Reinforcement driven information acquisition in nondeterministic environments, Technical Report, in preparation, Fakultät für Informatik, Technische Universität München.

P. Sollich. (1994) Query construction, entropy and generalization in neural network models. To appear in *Physical Review E.*

J. Subrahmonia, D. B. Cooper, and D. Keren. (1992) Practical reliable recognition of 2D and 3D objects using implicit polynomials and algebraic invariants. Technical Report LEMS-107, Division of Engineering, Brown University, Providence, RI.

R. Thisted. (1988) *Elements of Statistical Computing.* Chapman and Hall, NY.

S. Thrun and K. Möller. (1992) Active exploration in dynamic environments. In J. Moody et al., editors, *Advances in Neural Information Processing Systems 4.* Morgan Kaufmann, San Francisco, CA.

D. van Camp, T. Plate and G. Hinton. (1993) The Xerion Neural Network Simulator, Department of Computer Science, University of Toronto. For further information, send email to *xerion@cs.toronto.edu.*

S. Whitehead. (1991) A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, Department of Computer Science, Rochester University, Rochester, NY.

Appendix – System identification with neural networks and OED

System identification using OED has been successful on tasks where the parameters of the unknown system are explicit, but for a neural network model, system identification is problematic. The weights in the network can not be reasonably considered to represent real parameters of the unknown system being modeled, so there is no good interpretation of their "identity." A greater problem is the observation that unless the network is fortuitously structured to be exactly the correct size, there will be extra unconstrainable parameters in the form of unused weights, about which it will be impossible to gain information. Distinguishing between unconstrainable parameters (which we wish to delete or ignore) and underconstrained parameters (about which we wish to get more information) is an unsolved problem. Below, we review the derivation of the "D-optimality" criterion appropriate for system identification [Fedorov, 1972; MacKay, 1992], and briefly discuss experiments selecting D-optimal data.

When doing system identification with a neural network, we are interested in minimizing the covariance of the parameter estimates $\hat{w}$. For the purposes of optimization, it is convenient to express $\sigma_{\hat{w}}^2$ as a scalar. The most widely used scalar is the determinant $D = |\sigma_{\hat{w}}^2|$, which has an interpretation as the "volume" of parameter space encompassed by the variance (for other approaches see Atkinson and Donev [1992]).

The utility of querying at $\tilde{x}$, from a system identification viewpoint, may be expressed in terms of the expected change in the estimated value of D. The expected new value $\tilde{D}$ is

$$\tilde{D} = |\tilde{A}^{-1}| = \frac{S^2|A^{-1}|}{S^2 + g(\tilde{x})^T A^{-1} g(\tilde{x})} = \frac{S^2|\sigma_{\hat{w}}^2|}{S^2 + \sigma_{\hat{y}|\tilde{x}}^2}, \qquad (9)$$

which, by subtraction from the original estimate D gives

$$\Delta D|\tilde{x} \quad = \quad \frac{D\sigma_{\hat{y}|\tilde{x}}^2}{S^2 + \sigma_{\hat{y}|\tilde{x}}^2}. \qquad (10)$$

Equation 10 is maximized where $\sigma_{\hat{y}|\tilde{x}}^2$ is at a maximum, giving the intuitively pleasing interpretation that for system identification, parameter uncertainty is minimized by querying where our uncertainty is largest. Such queries are, in OED terminology, "D-optimal."

Our experiments using the above criterion to select training data had limited success. On regression problems such as the arm kinematics the learner performed poorly, attempting to select data at $x = \pm\infty$. These results are consistent with the comments at the beginning of this section, and with MacKay's observation that, for learning $X \rightarrow Y$ mappings, the system identification criterion may be the "right solution to the wrong problem" [MacKay, 1992]. The criterion addressed in Section 3, also mentioned by MacKay and explored in greater detail in this paper, appears to address the "right" problem.