6th International Workshop on Artificial Intelligence and Statistics, Fort Lauderdale, Florida, USA, January 4-7, 1997

Memory Based Stochastic Optimization for Validation and Tuning of Function Approximators *

Artur Dubrawski and Jeff Schneider
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

This paper focuses on the optimization of hyper-parameters for function approximators. We describe a kind of racing algorithm for continuous optimization problems that spends less time evaluating poor parameter settings and more time honing its estimates in the most promising regions of the parameter space. The algorithm is able to automatically optimize the parameters of a function approximator with less computation time. We demonstrate the algorithm on the problem of finding good parameters for a memory based learner and show the tradeoffs involved in choosing the right amount of computation to spend on each evaluation.

1 Introduction

Optimization of continuous-valued functions is a ubiquitous problem. Often, the optimization is done by iteratively selecting a point and evaluating it. If the evaluation is expensive, it is important to choose the points well. These problems show up commonly as "parameter tuning", which is something done frequently by engineers and scientists.

The case where the evaluations are perfectly accurate, or deterministic, is well studied and there are many algorithms ranging from well understood gradient-based methods to simulated annealing and poorly understood genetic algorithms.

Common approaches to the case where the evaluations are noisy, include genetic algorithms, response surface methods [2], and descendants of the method presented by Robbins and Monro [12][9]. These methods rely on the most recent data obtained to create a local model in the current region of interest. Memory based stochastic optimization [11] improves on this by using all the data to create a global nonlinear model, which is used in the selection of experiments. Dubrawski [6] demonstrated its use on the problem of tuning neural network hyper-parameters. We use the term *hyper-parameter* to refer to high-level settings such as learning rates, network architecture, or smoothing coefficients, as opposed to parameters that are set during training such as specific weights in a neural network.

We are interested in efficient search algorithms for finding good sets of function approximator hyper-parameters. We focus on the computational issues of finding a single best set of hyper-parameters by means of N-fold cross validation. Maron and Moore [10] showed how *racing* can be used to speed up parameter tuning for function approximators. The problem was to determine the best model from a discrete set of possible models. Racing begins by spending a little computation to get a rough estimate of the value of each model. As the algorithm proceeds, it spends more and more computation improving the estimates of the best models, but doesn't need to waste time with further evaluation of the obviously inferior models. The result was a significant reduction in the total amount of computation required to identify the best model compared to the naive algorithm of fully testing all models. In this paper, we investigate racing in the selection of models from a *continuous* valued space.

*This work has been partially supported by William J. Fulbright Foundation Research Grant to Artur Dubrawski.

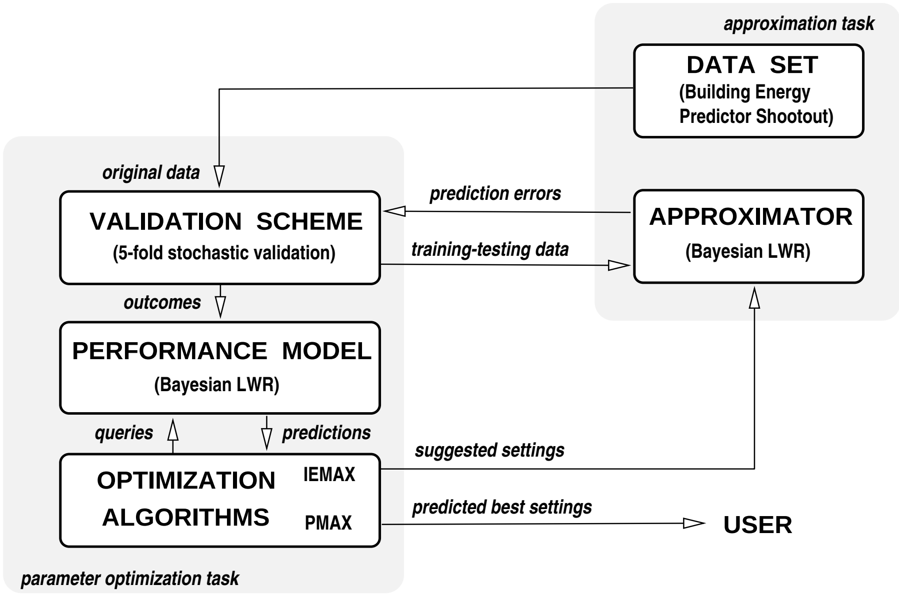

Figure 1: Stochastic validation and tuning applied to the cooling data approximation task. The validation scheme provides a way to evaluate the tuned approximator's performance over a given data set. The obtained outcomes are integrated into the Bayesian LWR model that provides predictions of the estimated performance and their confidences in response to the queries supplied by the optimization algorithms. These algorithms suggest the next hyper-parameters setting to try and provide the user with a prediction of the best setting detected so far.

The function we want to optimize is the mapping from model parameters to resulting prediction performance. Intuitively, it should be more difficult to optimize this function if the information we get about specific parameter settings is noisy. However, it may also be the case that noisy estimates are much cheaper to compute, and thus more queries can be made. We show results in which a kind of racing algorithm for continuous optimization problems spends less time on evaluations, thus making them more noisy, but allowing the algorithm to automatically optimize the parameters of a function approximator with less computation time. We demonstrate the algorithm on the problem of finding good parameters for a memory based learner and show the tradeoffs involved in choosing the right amount of computation to spend on each evaluation.

2 Stochastic Validation and Tuning

Parameter optimization proceeds by iteratively collecting evaluations of various parameter settings. A memory based model is used to represent the mapping from parameter setting to performance according to the data. Our optimization algorithm uses the model to select the next parameter setting. The new setting is evaluated with a resampling scheme and the process repeats. Each of these parts is described in this section.

2.1 Memory based model of a function approximator's performance

We use a form of locally weighted regression (LWR) [3, 1] called Bayesian locally weighted regression [11] to build a model from data. Constructing the model consists only of storing all the data points in an efficient structure (see [5] for more information on the techniques used). When a query, x_q, is made, each of the stored data points receives a weight $w_i = \exp(-\|x_i - x_q\|^2/K)$. K is the *kernel width* which controls the amount of localness in the regression. For plain LWR, the regression is done on these weighted points. However, the matrix inversion in the regression can be a problem when not enough

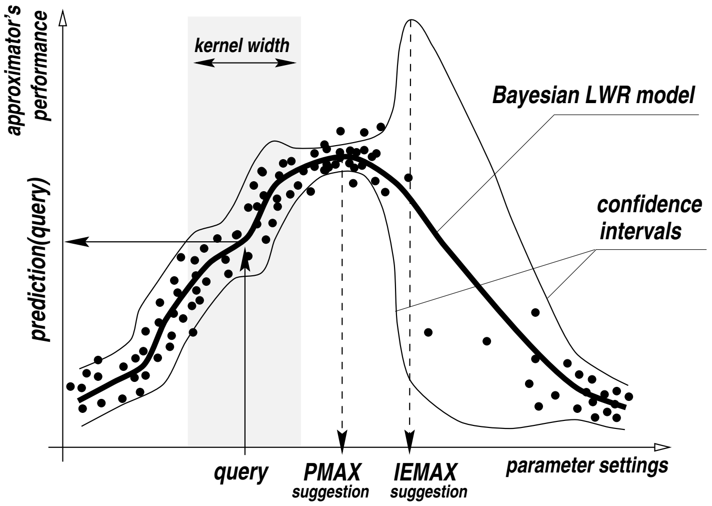

Figure 2: Memory-based learning by Bayesian locally weighted linear regression provides a way of modeling, exploring and optimizing performance of a stochastic process. Black dots represent experimentally obtained performance estimates, the thick line corresponds to the LWR model's predicted function, and the thin lines depict the model's confidence intervals. The location of the next experiment suggested by IEMAX corresponds to the location of the maximum of the upper confidence interval (or in a minimization task, the minimum of the lower confidence interval), while PMAX points out the current setting with the best predicted performance.

data is present in the locality of the query. For Bayesian LWR we assume a wide, weak gaussian prior on the coefficients of the regression model and a wide gamma prior on the inverse of the noise covariance. The result of a prediction is a t distribution on the output that is well defined and reasonable in the absence of data (see [11] and [4] for details).

2.2 The optimization algorithms

Each of the optimization algorithms described below makes its decision with information gained by querying a Bayesian LWR model constructed from all past experiments (Fig. 2).

- **PMAX** searches the current nonlinear model for its optimum and suggests it. This algorithm is not particularly good for stochastic optimization (although its intuition is commonly used in all kinds of iterative improvement algorithms). Its main use is to make a final recommendation after experimentation is complete and we use it to evaluate progress during optimization.
- **IEMAX** applies Kaelbling's Interval Estimation algorithm [7] in the continuous case using the Bayesian LWR confidence intervals. It suggests an experiment at the point whose upper confidence interval (95th percentile) is best. It explores aggressively throughout the entire search space in the hopes of finding the global optimum.

2.3 The resampling scheme

Evaluation of a learning algorithm from a single fold of the data into a training and testing set gives a cheap, but high variance estimate of the system performance. A lower variance estimate can be obtained with leave-one-out (l-o-o) cross validation, but it requires many train-and-test cycles, which can be expensive (LWR is a notable exception). A compromise solution is to use N-fold cross validation where the data is split into N subsets which are each used as testing sets to evaluate the performance of the function approximator trained on the other sets. Varying N gives a continuum of parameter evaluation methods.

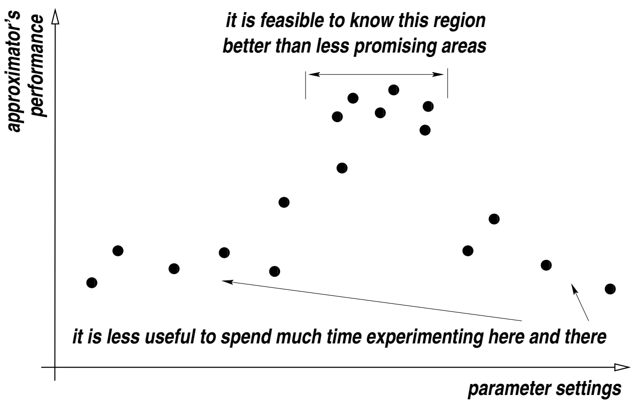

Figure 3: By using stochastic validation and memory-based optimization it is possible to quickly determine promising regions of the hyper-parameters space. During tuning, the amount of time spent for evaluating regions of low expectations is being reduced. Instead, more experiments are being performed along promising areas thus evaluations of increased confidence and accuracy are available where they are most desirable.

We can reduce the amount of time spent evaluating a parameter setting in exchange for higher variance in the resulting evaluation by reducing the number of testing subsets. Since our optimization scheme is specifically designed to handle noise, we can use noisy, inexpensive estimates to build a Bayesian memory-based performance model. This allows the stochastic optimization algorithm to quickly spot interesting regions in the parameter space and experiment there more often (Fig. 3).

3 Experiments

In the experiments described here we are looking for the best hyper-parameters for a Bayesian LWR algorithm trained to predict cooling load for a building, given solar radiation sensing (Fig. 1). Note that Bayesian LWR plays two roles in these experiments. First, it is at the heart of the stochastic optimization algorithm where it models the relationship between hyper-parameter settings and prediction performance. Second, the optimization of its own hyper-parameters for the cooling load data is the *target* of the optimizer in our experiments. The data set is taken from the Building Energy Predictor Shootout [8]. It contains four numeric input features which are potential predictors of a single numeric output attribute (the cooling load).

The performance of Bayesian LWR depends, given the data, on the particular setting of the algorithm's parameters: the kernel width and the weighted importance of each of the input features. In our data set, two of the input features are known to be important, and thus their weights are set to high constant values, while the remaining three parameters of the approximator (the kernel width and the weights on the other two input features) are adjusted by the optimizer.

Using Bayesian LWR for these experiments has the advantage that, unlike many other function approximator, leave-one-out cross validation can be done cheaply. Therefore, we use that as the final test of the optimizer's performance. The task can be stated as "search the space to find the parameter settings which minimize the l-o-o cross validation error over the entire training set."

3.1 Racing in continuous spaces

In [10] racing was shown on the problem of selecting the best function approximator from a discrete set of choices that were assumed to be unrelated. Small amounts of computation yielded noisy estimates of the value of each choice, and successively more computation was spent evaluating the most promising ones. Racing can happen in continuous spaces even if each parameter setting receives a full, perfect evaluation. The relationship between parameter settings and resulting performance is assumed to be

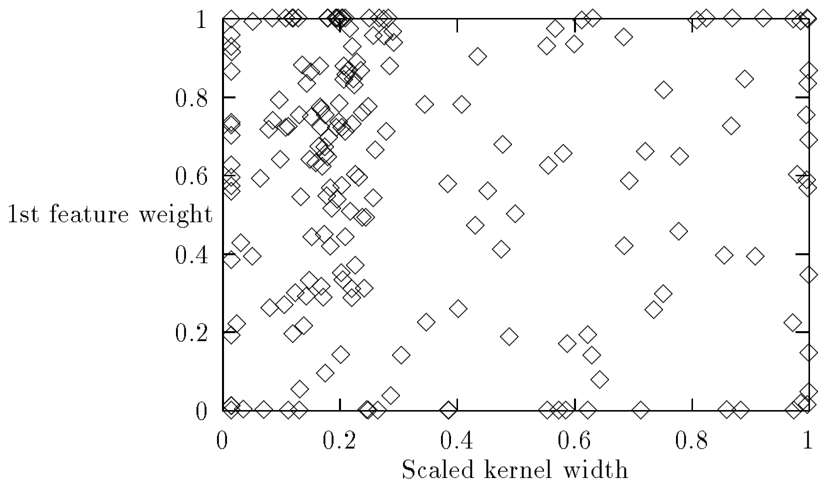

Figure 4: This is a 2-d projection (from 3-d) of the 200 experiments chosen by IEMAX during one optimization run. The concentration of points in the range of 0.1 to 0.25 for kernel width shows how IEMAX concentrates its experiments in the most promising regions. The plot also shows that more experiments were run with medium to high weights on the first feature. The best parameter settings are thought to be around a kernel width of 0.13 and a first feature weight of 0.9.

smooth, thus a full evaluation at a particular point gives information about the value of nearby points in parameter space. A smart experimentation algorithm will concentrate more samples in the more promising regions of the space.

To demonstrate racing in a continuous space we compare sampling the 3-d parameter space with a 6x6x6 grid vs. using IEMAX to choose 200 samples which will receive a full evaluation. The grid method does 216 samples in about the same time IEMAX can do 200. This is because the overhead of running IEMAX is slightly higher than the overhead for computing a sequence of grid points. The experiments chosen by IEMAX on one particular run are shown in fig. 4. This can be compared with what the evenly spaced grid would look like. Over 5 separate trials, the best setting found by the grid search had an average and standard deviation of 31.24 ± 1.11 in l-o-o cross validation error, while IEMAX obtained 26.54 ± 0.38.

3.2 Trading off evaluation accuracy for computational savings

In the previous experiments, each IEMAX choice received a complete leave one out evaluation. Although IEMAX easily outperformed a naive grid search method, it is possible to get further improvements by performing only a partial evaluation of each IEMAX selection. The resulting evaluation will be noisy, but our optimization algorithm is designed for noisy optimization and the savings in computation can be spent testing even more parameter settings. This adds a further level of racing. Not only do poor regions receive few experiments, but even those experiments need not be extensive. If the optimizer needs to get a better estimate on a particular setting, it will just ask for more experiments there (which is what it does). We also observe that although leave one out cross validation is cheap for Bayesian LWR, many other function approximators do not have this luxury, so in the following experiments we use multiple train and test folds during the experimentation process (but still use l-o-o to evaluate the optimizer's progress).

We randomly generate 100 training-testing data folds using N=5 (80% of the data for training, 20% for testing). A validation cycle is done by randomly choosing one of these folds and determining its test set error. IEMAX is used to suggest where to take experiments based on the performance model. We run these experiments using various numbers of validation cycles per experiment evaluation to see how the amount of resampling affects the optimization performance. The evaluation for an experiment is the average error over all the validation cycles done for that experiment.

.

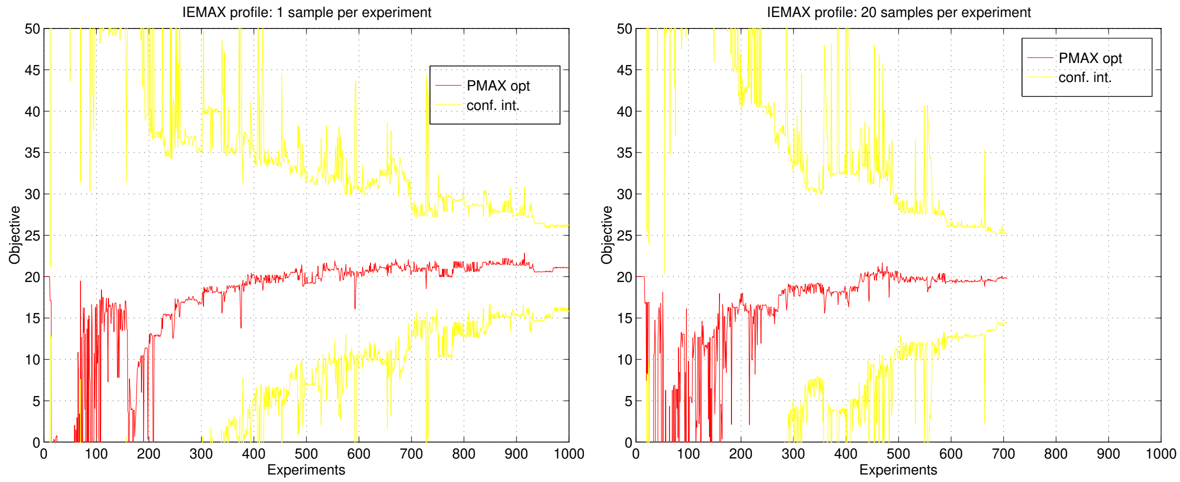

Figure 5: Profiles of the PMAX suggested settings during optimization with IEMAX using 1 (left) and 20 (right) evaluations per experiment. The graphs depict the development of the PMAX predicted error rates and confidence intervals on these predictions, at the parameter settings suggested by PMAX. The horizontal axis indicates the number of experiments performed and each graph spans 3 hours of optimization on a SparcStation 5

At each iteration, IEMAX is used in the optimization algorithm to suggest an experiment that will provide useful information. However, it does not necessarily suggest a setting that is *expected* to do well. We use PMAX for this purpose. The overall performance of the optimization process is evaluated by using PMAX to get the model's estimate of the best setting. At each iteration, we record the parameter setting suggested by PMAX, the estimated value of that setting, and its confidence. We also record the elapsed CPU time so experiments using different numbers of validations per experiment can be compared. Finally, full leave-one-out cross validation is performed off-line on PMAX's suggestions to evaluate them.

Fig. 5 shows example profiles of the performance of IEMAX optimization. Initially, PMAX's estimate of how well its chosen parameter settings will do is overly optimistic because it has little data and its confidence intervals are correspondingly wide. Near the end it is fairly confident in its recommendations. Note that 1000 experiments can be run when 1 sample is taken per experiment, but that number is reduced to 700 when 20 samples are taken per experiment. Also notice that even though the experiments return much noisier values when only one sample is taken per experiment, the confidence intervals at the selected best point are at least as small as in the 20 sample case. This happens because the extra experiments that can be run in the same amount of time are allocated to the important regions.

Table 1 summarizes the results of varying the number of samples per experiment for fixed amounts of optimization time. The best l-o-o scores are obtained by using 9 samples per experiment. When PMAX is not confident about its predictions, the difference between its estimate of the performance of a parameter setting and the actual leave-one-out cross validation performance is large. When it is more confident, the difference is small, but l-o-o generally yields a smaller value than the test set error which is as expected since more training data is available at each leave-one-out prediction.

4 Discussion

The experimental results show the benefits of using memory-based stochastic optimization to tune the hyper-parameters of function approximators. The stochastic optimization algorithm automatically allocates more computation to the most promising regions of the search space in order to find a better result and/or use less total computation.

Further improvements are possible when a less accurate performance estimate is accepted in exchange for reduced computational costs. The effect is to increase the granularity at which the optimization

Elapsed time	Samples per experiment	1	5	9	20	50
1 hour	True loo score of PMAX selection	24.2	24.7	24.7	20.9	23.2
	PMAX prediction	20.1	20.4	20.2	15.8	-1.9
	PMAX conf. int. width	20.3	24.9	25.0	48.1	108.8
2 hours	True loo score of PMAX selection	22.6	21.9	18.9	20.2	23.9
	PMAX prediction	21.6	20.9	19.7	20.6	18.3
	PMAX conf. int. width	14.5	13.7	11.7	21.5	56.5
3 hours	True loo score of PMAX selection	22.8	23.1	19.6	20.0	23.5
	PMAX prediction	21.0	21.1	20.5	19.8	18.1
	PMAX conf. int. width	10.3	13.2	11.9	10.7	36.2

Table 1: Summary of experimental results. PMAX prediction is the predicted performance (measured as a predictive error) of the parameter setting suggested by PMAX after the indicated amount of elapsed time. The confidence interval width shows how confident it is in its estimate. Loo score is the actual leave-one-out cross validation error at the parameter setting suggested by PMAX.

algorithm allocates its computational resources across the search space.

Testing with 50 samples per experiment yields a nearly deterministic evaluation of the parameter setting. At the other end of the spectrum, using only a single testing and training set gives a very noisy evaluation and exhibits racing in the sense that little computation is wasted on evaluations in bad regions and the bulk of it is spent honing the evaluation of good settings.

In table 1, the best results were obtained with a moderate number of samples per evaluation. There are two factors that trade off in the decision of how many samples to use in each evaluation. The first is the amount of time consumed per experiment. That time consists of a relatively fixed amount of overhead which comes from the execution of the stochastic optimization algorithm and a variable amount of computation which depends on how many samples are used. The second factor is the accuracy in the evaluations. As the number of samples becomes small, overhead in choosing which experiment to run dominates the time consumed in the experiment. The result is increasing noise in the evaluation without a corresponding reduction in the overall amount of time per experiment (thereby not giving an increase in the number of experiments that can be run). At the other end of the spectrum, as the number of testing and training sets becomes large, the evaluation is already quite accurate. Further increases consume more computation time per iteration without any corresponding improvements in evaluation accuracy.

The figures shown in table 1 are the preliminary results originally submitted with the extended abstract. The authors had hoped to provide a more complete set with the final paper. Unfortunately, the results of further experimentation were inconclusive in their support of our hypotheses about how changes in the number of validation samples per experiment effect overall optimization performance. There are several possible explanations for the difficulties and we are continuing our investigation of them. The most likely cause stems from the method by which the algorithm's progress is evaluated. We have observed PMAX selecting a mediocre parameter setting even though the information from the experiments done by IEMAX seems sufficient to make an excellent choice.

We have defined the task in our experiments as "find the parameter setting that produces the lowest leave-one-out cross validation score on the given training set." It should be noted that it is still possible to overfit the data while using this validation method, and the results of an optimization run should still be passed through a true test set as further insurance against overfitting.

This paper focuses on the optimization of parameters for function approximators. However, the methods reported here apply directly to any continuous-valued, noisy optimization problem where experiments are expensive. We are also working on applications of these algorithms to the optimization of industrial design and manufacturing processes.

References

[1] C. Atkeson. Using local models to control movement. In *Advances in Neural Information Processing Systems*, 1989.

[2] G. Box and N. Draper. *Empirical Model Building and Response Surfaces.* Wiley, 1987.

[3] W. Cleveland and S. Delvin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, pages 596–610, September 1988.

[4] M. DeGroot. *Optimal Statistical Decisions.* McGraw-Hill, 1970.

[5] K. Deng and A. Moore. Multiresolution instance-based learning. In *International Joint Conference on Artificial Intelligence*, 1995.

[6] A. Dubrawski. Memory-based stochastic optimization for automated tuning of neural network's high level parameters. In *4th International Symposium on Intelligent Robotic Systems*, 1996.

[7] L. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, June 1990.

[8] J. F. Kreider and J. S. Haberl. The great energy predictor shootout, June 1993.

[9] H. Kushner and D. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems.* Springer-Verlag, 1978.

[10] O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems (NIPS-6)*, 1993.

[11] A. Moore and J. Schneider. Memory based stochastic optimization. In *Advances in Neural Information Processing Systems (NIPS-8)*, 1995.

[12] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.