We must make automatic and habitual, as early as possible, as many useful actions as we can, and as carefully guard against the growing into ways that are likely to be disadvantageous.

– William James, 1899.

University of Alberta

AUTOMATIC STEP-SIZE ADAPTATION IN INCREMENTAL SUPERVISED LEARNING

by

Ashique Mahmood

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2010
Edmonton, Alberta

Examining Committee

Richard S. Sutton, Computing Science

Sirish L. Shah, Chemical and Materials Engineering

Dale Schuurmans, Computing Science

Dedicated to my parents

Abstract

Performance and stability of many iterative algorithms such as stochastic gradient descent largely depend on a fixed and scalar step-size parameter. Use of a fixed and scalar step-size value may lead to limited performance in many problems. We study several existing step-size adaptation algorithms in nonstationary, supervised learning problems using simulated and real-world data. We discover that effectiveness of the existing step-size adaptation algorithms requires tuning of a meta parameter across problems. We introduce a new algorithm—Autostep—by combining several new techniques with an existing algorithm, and demonstrate that it can effectively adapt a vector step-size parameter on all of our training and test problems without tuning its meta parameter across them. Autostep is the first step-size adaptation algorithm that can be used in widely different problems with the same setting of all of its parameters.

Acknowledgements

First and foremost, I would like to thank Dr. Richard Sutton for being such a great supervisor. He has assisted me through his ideas, scholarship, criticism and encouragement over the last one and a half years. Direction of this research is guided by his wisdom and philosophy. I have received great assistance and guidance from him in writing this thesis. His guidance not only helped me improve my skills as a researcher, but also enhanced my views on the philosophy of artificial intelligence and science.

This research is also carried out in collaboration with Dr. Thomas Degris. He provided me the programming framework for my initial experiments. His ideas and criticism helped me throughout this work. I thank Dr. Csaba Szepesvári for taking the time to answer my detailed questions and provide useful references regarding this work. Many thanks go to Hamid Reza Maei who helped me by spending time discussing this work. I thank Dr. Patrick Pilarski for helping me in this work through regular discussions. He thoroughly proofread the initial draft of this thesis. I would also like to thank Dr. Sirish Shah and Dr. Dale Schuurmans for participating on my thesis committee.

I am grateful to my father Rafique Ahmed, mother Jahanara Akter, brother Dr. Nasim Mahmood and sister Rafiqa Sharmin for their love and support. This work is indebted to their encouragement for higher education and learning. Finally, I am most grateful to my wife Farzana Yasmin (Liza) for supporting me throughout this work. She spent almost everyday listening to the small details of my struggles and failures. Her encouragements always made me strong and optimist. This work would have never been possible without her love and care.

Table of Contents

List of Figures

Chapter 1

The Problem of Step-Size Adaptation

Incremental updates of many iterative algorithms in machine learning, parameter estimation, system identification and adaptive control can be viewed as products of two factors. One of the factors can be thought as a step toward a direction close to the solution and the other can be thought as defining the size of this step. The latter is often known as the *step-size* parameter. Such a step-size parameter can either be a scalar or a vector or a matrix quantity. A manually-tuned, scalar constant is the most common.

Existence of a step-size parameter is often viewed as a disadvantage for such algorithms because it has to be tuned. Typically, performance and stability of the algorithms rely on the value of the step-size parameter. This brings the issue of how to set the step size properly for a specific problem. Finding a good value for the step size may require extensive manual tuning. Many practical problems involve changing environments and the solution in such problems may also change over time. In such problems, performance of algorithms with a constant step-size value, found through initial tuning, may diminish over time.

Nevertheless, many iterative algorithms with a step-size parameter are important and widely used in practice. Real-world problems can have high-volume, high-dimensional data. Iterative methods such as stochastic gradient descent are scalable to such large problems. Incremental updates are desired for these problems and many iterative algorithms with a step-size parameter are suitable for them.

This thesis explores algorithms of automatic step-size adaptation. In the first few chapters, we discuss several existing algorithms for step-size adaptation. A

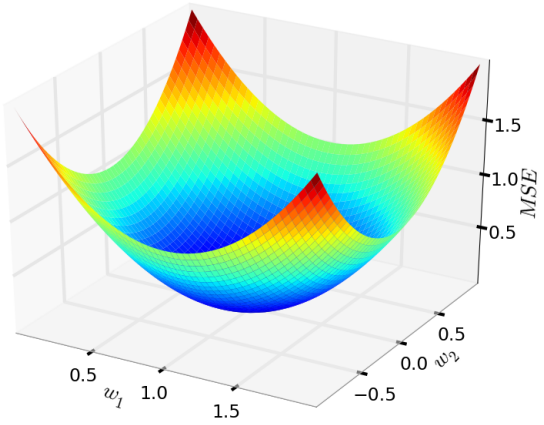

Figure 1.1: A spherical MSE function surface in weight-space. Target weight is at $\langle 1, 0 \rangle$. This function is quadratic in the weight space.

number of them are experimented on toy and real-world robot data. In later chapters, we introduce our proposed algorithm for automatic step-size adaptation.

1.1 The Least-Mean-Square (LMS) Setting

The LMS algorithm (Widrow and Hoff, 1960) is a concrete instance for studying step-size adaptation. It is the most widely used algorithm in supervised regression problems. Its typical use involves a fixed, scalar step-size parameter. For step-size adaptation, we use supervised regression problems and the LMS setting throughout this work.

In online supervised regression problems, a data sample $\boldsymbol{z}_t = \langle \boldsymbol{x}_t, y_t \rangle$ is observed at each time step t, where $\boldsymbol{x}_t \in \mathbb{R}^n$ is a vector of inputs and $y_t \in \mathbb{R}$ is a scalar, target output. In linear systems, the task of a learner is to predict the target output y_t as a linear combination of the inputs $\boldsymbol{w}_t^\top \boldsymbol{x}_t$, where $\boldsymbol{w}_t$ is the weight vector. This task can be formulated as the minimization of an error function, the Mean-Squared-Error (MSE) $\mathbb{E}_{\boldsymbol{z}_t}[\delta_t^2]$, where the sample error δ_t is the error in prediction, $y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t$, at time step t. MSE is a quadratic function on the weight vector $\boldsymbol{w}_t$, as shown in Figure 1.1. A learner updates the estimated weight vector $\boldsymbol{w}_t$ so that the solution or the target weight vector $\boldsymbol{w}^* = \min_{\boldsymbol{w}_t} \mathbb{E}_{\boldsymbol{z}_t}[\delta_t^2]$ could be found. Figure 1.1 shows the surface of one such function in weight space.

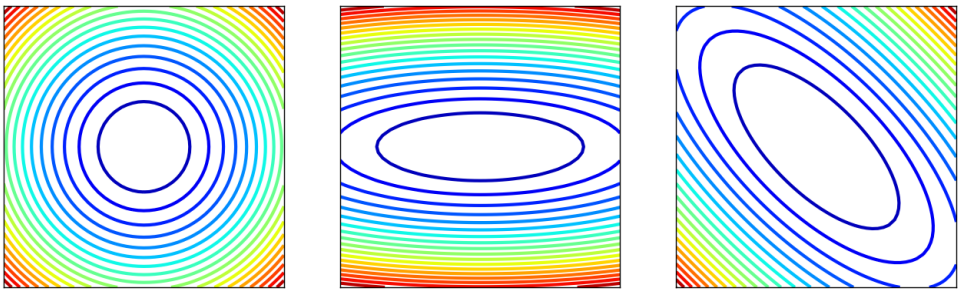

Figure 1.2: Contour lines of weight-space MSE functions with (LEFT) a spherical surface (MIDDLE) an ellipsoidal surface with the narrow valley parallel to one major axis and (RIGHT) an ellipsoidal surface with the narrow valley aligned diagonally across the major axes. The axes are the weight elements.

The LMS algorithm predicts the target output incrementally and online. LMS updates the weight vector $\boldsymbol{w}_t$ by following the opposite direction of the gradient of the sample squared error δ_t^2, as in the following:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla_{\boldsymbol{w}_t}\delta_t^2 \\
&= \boldsymbol{w}_t - \alpha\delta_t\nabla_{\boldsymbol{w}_t}\delta_t \\
&= \boldsymbol{w}_t - \alpha\delta_t\nabla_{\boldsymbol{w}_t}(y_t - \boldsymbol{w}_t^\top\boldsymbol{x}_t) \\
&= \boldsymbol{w}_t + \alpha\delta_t\boldsymbol{x}_t,
\end{aligned}$$

where $\alpha > 0$ is a fixed, scalar step-size parameter. The order of time and memory complexity of LMS for each iteration is linear on the size of the input vector.

If the target weight vector $\boldsymbol{w}^*$ is fixed, then with certain conditions on α the estimated weights $\boldsymbol{w}_t$ of the LMS algorithm converges in expectation to $\boldsymbol{w}^*$. Given that the conditions for stability are satisfied, then the larger the step size is, the quicker the LMS algorithm converges. On the other hand, asymptotic excess MSE for LMS is proportional to the value of the step-size parameter. It means, the higher the step-size value, the poorer will be the asymptotic performance. For stationary problems, starting with a high initial step-size value and decreasing it over time to zero can achieve both high convergence rate and better asymptotic performance. However, the target $\boldsymbol{w}^*$ may not be fixed and time-scheduling of the step-size parameter will perform poorly over time in such cases. A fixed step-size value is used in practice.

The operation of the LMS algorithm can be illustrated conveniently in the context of the weight space MSE-function surface. The MSE surface in the weight space can take a form of a spherical bowl, as shown in Figure 1.1. This MSE function was calculated from 3000 randomly generated inputs and outputs. Two dimensional input vectors $\boldsymbol{x}$ were drawn independently and randomly from a normal distribution with zero mean and unit variance. The target outputs y were calculated as a linear combination of the inputs: $\boldsymbol{w}^{*\top}\boldsymbol{x}$, where the target weight vector, $\boldsymbol{w}^* = \langle 1, 0 \rangle$. The contour lines of the same function is shown in the left panel of Figure 1.2.

Estimating the solution by following the opposite direction of gradient is a good idea in this kind of surfaces, because the gradient direction always points toward the solution in such surfaces and a small step in that direction reduces the MSE. Gradient descent is the steepest-descent direction, i.e., error function reduces fastest at one time step along the gradient direction (Bazaraa and Shetty, 1979). The sample gradient $\nabla_{\boldsymbol{w}_t}\delta_t^2$ in LMS is a noisy gradient estimate on this surface at $\boldsymbol{w}_t$. A scalar step-size parameter defines how much the weight update is scaled toward this noisy gradient-descent direction. Typically, the LMS algorithm with a small value for a scalar step-size parameter works well in this kind of problems.

Gradient direction with a scalar step-size value does not work out well in all problems. An example of difficult problems for LMS is where the MSE function is ellipsoidal, as shown in the middle panel of Figure 1.2. This function was found by using the data samples for the spherical function, shown in the left of the same figure and multiplying the second component of all the input vectors by three. The variance of the second input is then nine, while the variance of the first input is one. Due to different variances in inputs, the surface has become an ellipsoid. Here the narrow valley is parallel to the X axis. The steepness is more along Y axis than along the X axis. Performance of the LMS algorithm is slow in problems with ellipsoidal MSE function. Although, a small step toward the gradient-descent direction can still reduce the error, this direction on average does not point toward the solution. A scalar step size can only scale the amount of the step toward the gradient-descent direction. The LMS algorithm, which uses a noisy gadient-descent direction and a

scalar step-size parameter, takes zigzagging steps toward the minimum of the surface.

By scaling the gradient elements along each direction separately using a vector step-size parameter, an effect similar to the case of spherical surface can be achieved, i.e., the update direction will point toward the target in expectation. Using a vector step-size parameter requires the same order of complexity as LMS, because the time and memory complexity of using a vector is only linear on the size of the input vector. This clearly demonstrates the advantage of having a vector step-size parameter in LMS without increasing the order of its time and memory complexity.

In general, for a proper scaling of the gradient vector in ellipsoidal problems, a matrix step-size parameter is needed. An instance where scaling through a vector step-size parameter can be inadequate is when the narrow valley of an ellipsoidal MSE is diagonally aligned across the axes. Such an example is shown in the right panel of Figure 1.2. Here, both inputs have unit variance, but there is a non-zero covariance of 0.7 between them. A matrix step size is needed to scale the gradient elements properly in such cases.

Use of a matrix requires quadratic order of time and memory complexity on the size of the input vector. Therefore, using a matrix step-size parameter would increase the order of the complexity of LMS. There can be many applications, where increasing the order of time or memory complexity is not desirable. Therefore, a vector step-size parameter would be preferable to a matrix in many large problems.

In problems where the number of inputs is very large, i.e., a million, manual tuning of the numerous elements of a vector step size can be impractical. In this thesis we are interested in automatically adapting a vector step-size parameter of the LMS algorithm.

1.2 Nonstationarity

The limitation of using a scalar step-size parameter can also be observed in nonstationary problems. A source of nonstationarity is when the target solution $\boldsymbol{w}^*$

changes over time. Such nonstationarity can occur in dynamic systems with changing environments. Limited function approximation may also introduce such nonstationarity.

In nonstationary problems, the step-size value needs to be adequately high in order to track the solution as fast as possible. On the other hand, high step-size values give high asymptotic excess MSE in stationary problems and a low step-size value is desirable in such cases. A contrasting demand is posed on a scalar step-size parameter when there are both stationary and nonstationary target weights in the same problem. Due to having a scalar step size, performance of the LMS algorithm is limited in such problems.

A per-input or vector step-size parameter can improve performance substantially in problems with both stationary and non-staitonary target weights. The step-size elements corresponding to the stationary target weights can be set to small values, while for nonstationary target weights, to adequately high values.

Advantage of using a vector step-size parameter can be best demonstrated in such nonstationary settings, i.e., where some target weights are stationary while some others are nonstationary. Nonstationary problems are also relevant in real-world applications. In this thesis we explore nonstationary problems for studying step-size adaptation.

1.3 Feature Relevance and Learning Bias

Using per-input step-size parameters can be connected with the concept of feature relevance. Inputs are often thought as features of learning. A good bias of learning can be achieved by selecting an appropriate set of features. Finding feature relevance is one such criteria for feature selection. An input with a corresponding non-zero positive step-size value can be interpreted as a relevant feature. On the other hand, an input with zero step-size value does not contribute to the estimation of weights and hence can be thought as an irrelevant feature. Good values of step-size parameters, therefore, correspond to appropriate feature relevance and thus form a good learning bias.

A widely used technique for finding a good learning bias is using cross validation. The technique of cross validation is used to find appropriate parameter values so that overfitting is avoided. In k-fold cross validation, the data set is divided in k partitions. One partition is held out as a validation set and rest of the partitions are used as a training set. For each choice of a system parameter, the learner is trained on the training set and performance is validated over the validation set. Validation error over k different validation sets is then averaged to find the overall performance. Values of the system parameters are chosen from this process so that average validation error is minimum. An extreme form of this setting is hold-one-out cross validation, where only one data sample is held out for the validation, each time. Such process can be used to find the appropriate values of the step-size parameter.

For conducting cross validation, all the data are needed together at the same time. Therefore, this form of cross validation is offline and parameter tuning through this process is an extensive form of search over the parameter space.

A step-size adaptation algorithm that iteratively updates the values of per-input step-size parameters and validates the updated step-size value on a new data sample at each step can be thought as an online form of hold-one-out cross validation. This can update the parameter values incrementally and online without an extensive search. Such a view is also held in a work by Sutton (1992a), where he proposed one such algorithm for step-size adaptation and interprets it as an incremental form of hold-one-out cross validation.

Viewing step-size adaptation as an incremental and online cross validation technique for finding good learning bias clearly establishes the need for such methods.

1.4 Desiderata of Step-Size Adaptation Algorithms

Here we establish a number of desirable criteria for step-size adaptation based on our discussions of nonstationarity, learning bias and feature relevance. This thesis explores other works in the context of these criteria. For our proposed algorithms, these criteria are considered as the desirable goals of step-size adaptation.

Performance: In nonstationary problems, fast tracking is important for superior performance. A primary goal of step-size adaptation is to track the solution as fast as possible by adapting to an adequately high step-size value. This goal is suggested in many prior works such as those by Sompolinsky et al. (1995), Murata et al. (1996) and George and Powell (2006). On the other hand, in problems with stationary solutions, a high step-size value deteriorates asymptotic performance (Haykin, 2001). Step-size values are required to diminish over time so that asymptotic error is small.

In practical problems with large number of inputs, it is more common to have both stationary and nonstationary target weights at the same time. The goal of step-size adaptation in such a situation is to deal with both stationary and nonstationary elements at the same time so that good performance can be achieved.

Linear Complexity: In many real-time applications, data grows fast and abundantly, imposing a time limit on the computation for each sample. Online, incremental methods that estimate a solution as fast as possible have advantage over computationally expensive methods in real-time systems. Methods with linear time and memory complexity, thus, have clear advantage over higher order methods. The LMS algorithm, which we consider as the base system in this thesis, also requires linear time complexity. In adapting the step-size parameter of LMS, our goal is to keep the order of time and memory complexity as much as that of LMS.

Vector Step Size: In problems where target weight elements can be both stationary and nonstationary, a vector step size is appropriate in dealing with such weights separately. Adaptation of a vector step-size parameter can serve as a feature selection procedure and form a good learning bias. A vector step-size parameter also has an advantage over a scalar one when the error surface has largely different curvatures in different directions. Sutton (1992a) also viewed adaptation of a vector step-size parameter as a useful goal for nonstationary problems.

In large-scale, real-world problems, where both performance and computation time is crucial, use of a vector step-size parameter is preferable to a matrix step-size parameter. A matrix step-size parameter will increase the order of the complexity of LMS or such first order methods, by requiring quadratic computation instead of linear. Using a vector step-size parameter, in that sense, is the best we can do without increasing the order of the complexity of LMS.

Automatic Adaptation: Given that an incremental algorithm uses a vector step-size, manually setting the elements of a large vector step-size parameter for a high-dimensional, nonstationary problem is extremely difficult. It is desirable to compute appropriate step-size values without an extensive parameter tuning. Our goal of step-size adaptation is to achieve one that is completely automatic in its procedure.

These goals are relevant for many practical problems. If a step-size adaptation algorithm fulfilling these goals is applied in a learning algorithm, performance can be substantially improved and other advantages such as feature selection can also be gained.

1.5 Contributions

This thesis explores the ideas of step-size adaptation in LMS setting. We explore several existing algorithms that adapt the step-size parameter. We find that such algorithms are limited by requiring manual tuning of other parameters. This thesis focuses on introducing techniques in order to develop a completely automated step-size adaptation algorithm.

Following are the contributions of the thesis:

- We empirically analyze and compare the existing step-size adaptation algorithms and demonstrate that applicability of these algorithms are limited by their requirement of manual tuning of one or more parameters. Many algorithms use a gradient-based rule for adapting the step-size parameter. We

demonstrate that the effectiveness of these algorithms largely depends on the choice of a meta-step-size parameter that controls the speed of the adaptation. The best choice of the meta-step-size parameter is found to be problem dependent, varying in orders of magnitude between problems. We also show that the range of best values of this parameter for a specific problem is narrow for most of these algorithms.

- We devise a new step-size adaptation algorithm for linear supervised regression learning that incrementally and effectively adapt step-size without requiring any prior knowledge of the system or an extensive parameter tuning of any kind. Performance of our algorithm is qualitatively less sensitive to the choice of the meta-step-size parameter. We demonstrate that our algorithm can effectively adapt the step-size parameter on all of our toy and real test problems using the same setting of its parameters.

1.6 Application Areas

Adaptation of the step-size parameter has usefulness in many practical problems. Examples include 3D hand tracking (Bray et al., 2004), learning sensorimotor transformations of human and robot arm movements (Vijayakumar and Schaal, 1998), aircraft-navigation trajectory tracking (Bousson, 2007) and learning the shape of a receptive field in weighted regression (Schaal and Atkeson, 1997). In the following, we list some of the problem domains where an automatic step-size adaptation can be useful.

Problems with High-Volume Data: In many practical problems involving real-time sensor signals, web-data, video, audio, image, etc., the size of data can be extremely large and ever-growing. Making use of such big data sets is a difficult challenge and classical optimization methods that perform passes over a data set before providing a solution can be very expensive and undesirable. An automatic and incremental step-size adaptation that can estimate a solution and update the step-size parameter as soon as evaluating one data sample is suitable for such applications.

On the contrary, problems where data is scarce and an incremental algorithm is not a method of choice, step-size adaptation does not have a strong application either.

Problems with Hi-Dimensional Data: In many problems, the input dimensions of a sample data can be as large as millions. Examples include—neural networks with a large number of hidden units, function approximation techniques in reinforcement learning algorithms that produce a large number of features from a relatively small number of inputs (Sutton and Barto, 1998) and systems with large number of sensor fields. High dimensions enable a system to capture a wide range of data patterns. Methods that require high order computation time on the number of input dimension can be prohibitive. A first order incremental algorithm is scalable in such problems. Using a vector step-size for these algorithms would require tuning of a large number of step-size elements, as much as the number of input dimensions, if an adaptive algorithm is not applied. Using a step-size adaptation algorithm have definite advantage in such problems.

On the other hand, problems where extensive search of parameters is a feasible option, step-size adaptation may not be a preferable alternative. These include problems with a small number of input dimensions. However, low-dimensional data does not necessarily mean that parameter search is feasible. One example is nonstationary problems that we describe next.

Online Learning in Nonstationary Problems: In practical problems, it is common to have the distribution of data changing over time. Many classical methods consider a batch of data together and hence can be ineffective for such problems. Extensive parameter search for these problems is infeasible even for low-dimensional data when they contain nonstationarity, because parameter search is now required not only across the space but also across the time. Many incremental algorithms are preferred for online learning and step-size adaptation for these algorithms is an automated alternative to the extensive parameter search.

Lifelong learning: Besides all the areas of applications mentioned above, step-size adaptation finds its best use in a lifelong learning setting where a system runs for a long time without any human intervention. Examples include robotic systems that are required to perform similarly to living animals or robots performing a task in a remote place where human intervention is not a choice. Such systems typically compose of several other sub-problems that might be thought as concurrently running optimizations. Manual tuning of system parameters for each of these sub-problems, specially when their best values change over the time, is not an option for these systems. The idea of automatic step-size adaptation can be extended to the automatic adaptation of any kind of bias due to system parameters and thus is an appropriate solution for these systems.

An automatic step-size adaptation algorithm can find its best use in large-scale online learning problems. This thesis introduces one such algorithm for online supervised regression by adapting a vector step-size parameter in LMS setting. Our algorithm is the first such kind that does not require extensive parameter tuning in adapting the step size.

Chapter 2

Step-Size Adaptation Algorithms

In this chapter we discuss some of the basic ideas of step-size adaptation and describe several existing algorithms based on these ideas. Most of these algorithms are introduced for fast tracking in nonstationary problems.

We categorize these algorithms into two groups: meta-descent algorithms and running-average-based algorithms. Meta-descent algorithms iteratively update the step-size parameter based on a gradient-descent rule. A gradient-descent rule is typically used for updating the weights of the learning system. Therefore, when a gradient-descent rule is also applied to learn the step-size parameter, we refer to it as a *meta-descent* algorithm. There are also other forms of step-size adaptation where the step-size parameter is iteratively updated as a running average of some observable measure. The particular measure used for the average depends on the specific algorithm.

Many of the existing step-size adaptation algorithms are proposed for a general stochastic gradient descent setting. They can be easily extensible to linear supervised learning setting by considering them as stochastic gradient descent on the sample squared error. In such cases, we first describe the algorithm in their general setting and then discuss them in a linear supervised learning setting.

In some of these algorithms, it is proposed to update the step-size parameter before updating the weight, while for some others, it is proposed to update the step size after. For some of them, the order is not mentioned at all. Here in the description of all the algorithms in the linear supervised learning setting, the step-size update is used before the weights in order to maintain a clear comparison among all

the algorithms. These descriptions of algorithms will be used for experiments later in this thesis. We made sure that such altering of the update orders did not harm the performance of any of the algorithms.

Finally, at the end of this chapter, we point out the algorithms that are interesting according to our goals of step-size adaptation, set in Chapter 1.

2.1 Step-Size Adaptation using Meta Descent

The idea of meta descent is to adapt the step-size parameter based on a gradient-descent rule. Work by Sutton (1981) was among the first to propose such an idea. In meta descent, the step-size parameter is adapted by following the opposite direction of the gradient in the step-size space surface of a loss function:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \frac{1}{2}\theta \nabla_{\boldsymbol{\alpha}_t} l(\boldsymbol{\alpha}_t), \tag{2.1}$$

where $\theta > 0$ is a fixed, scalar meta-step-size parameter, $\boldsymbol{\alpha}_t$ is a vector step-size parameter that is being adapted, $l(\boldsymbol{\alpha}_t)$ is a loss function and $\nabla_{\boldsymbol{\alpha}_t} l(\boldsymbol{\alpha}_t)$ is a meta-gradient term. This loss function $l(\boldsymbol{\alpha}_t)$ can be the same loss function that the weight update rule is minimizing, which is the sample squared error for the LMS algorithm.

Following is the weight update rule of the LMS algorithm with a vector step-size parameter $\boldsymbol{\alpha}_t$:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t),$$

where the operator $\circ$ is the element-wise vector product.

A vector step-size parameter is useful when the error surface have different curvatures in different directions, as illustrated in Chapter 1. Such a vector step-size parameter is also useful in nonstationary problems where there are irrelevant inputs. The weight update $\Delta \boldsymbol{w}_t = \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$ is not any more a steepest descent direction. However, positive values of the elements of $\boldsymbol{\alpha}_{t+1}$ ensure that $\Delta \boldsymbol{w}_t$ follows a descent direction.

We define the gradient of a vector $\boldsymbol{p}$ with respect to a vector $\boldsymbol{r}$ as a matrix $\nabla_{\boldsymbol{r}}\boldsymbol{p}^\top$, where—

$$\left\{\nabla_{\boldsymbol{r}}\boldsymbol{p}^\top\right\}_{i,j} = \frac{\partial p_j}{\partial r_i}.$$

Here, $\{M\}_{i,j}$ is the ith row and jth column of a matrix M and v_i is the ith element of a vector $\boldsymbol{v}$.

It is worthy to note that, the gradient of an element-wise vector product $\boldsymbol{p} \circ \boldsymbol{q}$ with respect to a vector $\boldsymbol{r}$ can be written as follows:

$$\nabla_{\boldsymbol{r}}\left[\boldsymbol{p} \circ \boldsymbol{q}\right]^\top = \left(\nabla_{\boldsymbol{r}}\boldsymbol{p}^\top\right)\mathrm{Diag}\left(\boldsymbol{q}\right) + \left(\nabla_{\boldsymbol{r}}\boldsymbol{q}^\top\right)\mathrm{Diag}\left(\boldsymbol{p}\right),$$

where the operator $\mathrm{Diag}\left(.\right)$ takes a vector and produces a diagonal matrix: $\left\{\mathrm{Diag}\left(\boldsymbol{v}\right)\right\}_{i,i} = v_i$ and $\left\{\mathrm{Diag}\left(\boldsymbol{v}\right)\right\}_{i,j} = 0$, where $i \neq j$.

If we consider the loss function minimized in the step-size update rule to be the sample squared error δ_t^2, then Equation (2.1) can be manipulated as in the following:

$$\begin{aligned}
\boldsymbol{\alpha}_{t+1} &= \boldsymbol{\alpha}_t - \frac{1}{2}\theta\nabla_{\boldsymbol{\alpha}_t}\delta_t^2 \\
&= \boldsymbol{\alpha}_t - \frac{1}{2}\theta\left(\nabla_{\boldsymbol{\alpha}_t}\boldsymbol{w}_t^\top\right)\left(\nabla_{\boldsymbol{w}_t}\delta_t^2\right) \\
&= \boldsymbol{\alpha}_t + \theta\left(\nabla_{\boldsymbol{\alpha}_t}\left[\boldsymbol{w}_{t-1} + \boldsymbol{\alpha}_t \circ \left(\delta_{t-1}\boldsymbol{x}_{t-1}\right)\right]^\top\right)\delta_t\boldsymbol{x}_t \\
&= \boldsymbol{\alpha}_t + \theta\left(\nabla_{\boldsymbol{\alpha}_t}\left[\boldsymbol{\alpha}_t \circ \left(\delta_{t-1}\boldsymbol{x}_{t-1}\right)\right]^\top\right)\delta_t\boldsymbol{x}_t \\
&= \boldsymbol{\alpha}_t + \theta\,\mathrm{Diag}\left(\delta_{t-1}\boldsymbol{x}_{t-1}\right)\left(\delta_t\boldsymbol{x}_t\right) \\
&= \boldsymbol{\alpha}_t + \theta\left(\delta_t\boldsymbol{x}_t\right) \circ \left(\delta_{t-1}\boldsymbol{x}_{t-1}\right) \\
&= \boldsymbol{\alpha}_t + \theta\left(-\delta_t\boldsymbol{x}_t\right) \circ \left(-\delta_{t-1}\boldsymbol{x}_{t-1}\right) \\
&= \boldsymbol{\alpha}_t + \theta\left(\nabla_{\boldsymbol{w}_t}\delta_t^2\right) \circ \left(\nabla_{\boldsymbol{w}_{t-1}}\delta_{t-1}^2\right). \qquad (2.2)
\end{aligned}$$

Rules similar to this have been proposed several times in the literature, e.g., by Sutton (1981), Jacobs (1988) and Almeida et al. (1998). Jacobs (1988) refers to this as the delta-delta learning rule, where delta refers to both $\nabla_{\boldsymbol{w}_t}\delta_t^2$ and $\nabla_{\boldsymbol{w}_{t-1}}\delta_{t-1}^2$,

the gradients of the sample squared error with respect to the weight vector in two successive time steps.

This method can enhance the performance of LMS. When a gradient element has the same signs on consecutive time steps, it is frequently the case that the corresponding weight in the error surface continues to slope down in the same direction. It can occur in a case, where the curvature along this direction is small or the minimum at that direction is moving. For both cases, delta-delta rule accelerates the speed of learning by increasing the step-size element in that direction according to Equation (2.2). For stationary cases, it helps reach the minimum faster and for nonstationary cases, it tracks the solution faster.

When a gradient element has opposite signs for a number of time steps, it is more likely that the estimate is frequently overshooting the minimum in this direction. Such frequent overshooting results in oscillation around the solution and hence, large excess Mean Squared Error (MSE). Delta-delta rule improves performance in such a case by reducing the step-size element along that direction.

Delta-delta rule, however, has a number of problems. This form of step-size update rule is highly stochastic in nature as both of the gradient terms in the element-wise product of the update in Equation (2.2) are noisy. The fact that the step-size parameter does not only influence the current weight in the next time step but also affects all future ones is not considered in this rule.

There are also several cases where delta-delta rule can be ineffective, for example, problems where inputs are binary. An extreme case is when the inputs frequently alternates between zero and one at every time step. The elements of the step-size update, $(\nabla_{\boldsymbol{w}_t}\delta_t^2) \circ \left(\nabla_{\boldsymbol{w}_{t-1}}\delta_{t-1}^2\right)$, will always remain zero in this case and delta-delta rule will not be able to adapt step-sizes at all.

Moreover, step-size values in delta-delta rule can go negative and imposing some lower bound is needed to keep it positive at every time step. The additive form of this step-size update rule can also take a long time until the step-size parameter is adjusted to an appropriate value.

This stochasticity can be reduced by replacing the gradient term from the immediate past, $\nabla_{\boldsymbol{w}_{t-1}}\delta_{t-1}^2$ by a trace or running average of the past gradients. One

such technique for batch learning is proposed by Jacobs (1988). He referred it as delta-bar-delta rule, where delta bar refers to the average of the past gradients.

2.2 Incremental Delta-Bar-Delta (IDBD)

Sutton (1992a) proposed Incremental Delta-Bar-Delta (IDBD) that considers the delta-bar-delta idea by Jacobs (1988) for online, incremental setting. However, it does not use the delta-bar-delta idea by merely taking the running average of past gradients. The effect of the step size on all future weights is taken into consideration by considering the effect of all past step-size values on the current weights. An incremental update of an extra variable maintains the propagation of this effect from one time step to the next.

IDBD avoids the problem of negative step-size values and slow additive updates by expressing the vector step-size parameter as an exponential function of another vector parameter $\boldsymbol{\beta}_t$:

$$\boldsymbol{\alpha}_t = e^{\boldsymbol{\beta}_t}.$$

This vector parameter $\boldsymbol{\beta}_t$ is updated by approximating the gradient of the sample squared error with respect to $\boldsymbol{\beta}$:

$$\boldsymbol{\beta}_{t+1} \approx \boldsymbol{\beta}_t - \frac{1}{2}\theta \nabla_{\boldsymbol{\beta}} \delta_t^2,$$

where gradient with respect to $\boldsymbol{\beta}$ should be interpreted as an infinitesimal change in $\boldsymbol{\beta}_k$ at all time steps k, as in the following:

$$\nabla_{\boldsymbol{\beta}} \delta_t^2 = \sum_{k=0}^{t} \nabla_{\boldsymbol{\beta}_k} \delta_t^2.$$

The gradient term $\nabla_{\boldsymbol{\beta}} \delta_t^2$ can be expressed as:

$$\begin{aligned} \nabla_{\boldsymbol{\beta}} \delta_t^2 &= \left(\nabla_{\boldsymbol{\beta}} \boldsymbol{w}_t^\top\right) \nabla_{\boldsymbol{w}_t} \delta_t^2 \\ &= -2G_t \left(\delta_t \boldsymbol{x}_t\right), \end{aligned}$$

where $G_t = \nabla_{\boldsymbol{\beta}} \boldsymbol{w}_t^\top$ is a matrix.

As the computation of a matrix requires quadratic time and memory complexity, this matrix is diagonally approximated by considering the non-diagonal elements to be zero. IDBD updates $\boldsymbol{\beta}_t$ as:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \theta \delta_t \boldsymbol{x}_t \circ \boldsymbol{h}_t,$$

where the vector $\boldsymbol{h}_t$, approximates the diagonal elements of the matrix G_t.

To diagonally approximate using the vector $\boldsymbol{h}_t$, first we manipulate G_t as follows:

$$\begin{aligned}
G_{t+1} &= \nabla_{\boldsymbol{\beta}} \boldsymbol{w}_{t+1}^\top \\
&= \nabla_{\boldsymbol{\beta}} \left[\boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)\right]^\top \\
&= \nabla_{\boldsymbol{\beta}} \boldsymbol{w}_t^\top + (\nabla_{\boldsymbol{\beta}} \boldsymbol{\alpha}_{t+1}) \operatorname{Diag}(\delta_t \boldsymbol{x}_t) + (\nabla_{\boldsymbol{\beta}} \delta_t) \boldsymbol{x}_t^\top \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \\
&= G_t + \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \operatorname{Diag}(\delta_t \boldsymbol{x}_t) + \nabla_{\boldsymbol{\beta}} \boldsymbol{w}_t^\top (\nabla_{\boldsymbol{w}_t} \delta_t) \boldsymbol{x}_t^\top \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}).
\end{aligned}$$

Notice that $\nabla_{\boldsymbol{w}_t} \delta_t$ can be written simply as

$$\nabla_{\boldsymbol{w}_t} \delta_t = \nabla_{\boldsymbol{w}_t} \left[y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right] = -\boldsymbol{x}_t.$$

Therefore, we can write the update for G_t as

$$\begin{aligned}
G_{t+1} &= G_t + \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \operatorname{Diag}(\delta_t \boldsymbol{x}_t) - \nabla_{\boldsymbol{\beta}} \boldsymbol{w}_t^\top \boldsymbol{x}_t \boldsymbol{x}_t^\top \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \\
&= G_t + \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \operatorname{Diag}(\delta_t \boldsymbol{x}_t) - G_t \boldsymbol{x}_t \boldsymbol{x}_t^\top \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \\
&= G_t - G_t \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \boldsymbol{x}_t \boldsymbol{x}_t^\top + \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \operatorname{Diag}(\delta_t \boldsymbol{x}_t). \qquad (2.3)
\end{aligned}$$

Now, the update of $\boldsymbol{h}_t$ for IDBD can be found by diagonally approximating the update of G_t as

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t - \boldsymbol{h}_t \circ \boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t \circ \boldsymbol{x}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t), \qquad (2.4)$$

where the non-diagonal terms of G_t are considered zero. The vector consisting the diagonal terms of $G_t \operatorname{Diag}(\boldsymbol{\alpha}_{t+1}) \boldsymbol{x}_t \boldsymbol{x}_t^\top$ is $\boldsymbol{h}_t \circ \boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t \circ \boldsymbol{x}_t$.

The update rules of IDBD for each iteration in vector element form is as follows:

$$
\begin{aligned}
\beta_{i,t+1} &= \beta_{i,t} + \theta \delta_t x_{i,t} h_{i,t}, \\
\alpha_{i,t+1} &= e^{\beta_{i,t+1}}, \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\
h_{i,t+1} &= h_{i,t} \left(1 - \alpha_{i,t+1} x_{i,t}^2\right)^+ + \alpha_{i,t+1} \delta_t x_{i,t},
\end{aligned}
$$

where $[v]^+$ is v, if $v > 0$, otherwise 0 to ensure that $(1 - \alpha_{i,t+1} x_{i,t}^2)$ is never negative. The additional per-input memory variable h_i can be interpreted as a trace of $\delta_t x_{i,t}$, the gradient of the sample squared error with respect to w_i. The term $\delta_t x_{i,t} h_{i,t}$ is sometimes referred as the meta-gradient term of IDBD in this thesis.

The update of IDBD remains less affected by the stochasticity than delta-delta rule because the gradient term from the previous time step $\delta_{t-1} \boldsymbol{x}_{t-1}$ is now replaced by $\boldsymbol{h}_t$, the trace of past gradients.

The operation of IDBD can be explained in a similar way as the operation of the delta-delta rule, i.e., it increases the step-size value when the estimate is frequently heading to the solution from the same side of the surface and decreases it when the weight estimate is likely to be oscillating around the target. A step-size element α_i increases when $\delta_t x_{i,t} h_{i,t}$ has a positive sign, that is when the current gradient element $\delta_t x_{i,t}$ is in the same direction as the trace of past gradient elements h_i. Such a case frequently corresponds to a weight element, the estimated value of which is still far from the target weight. For a nonstationary target weight element, this kind of step-size update will keep the corresponding step size always high so that tracking of the corresponding target weight element is possible.

A step-size element α_i decreases when $\delta_t x_{i,t}$ and $h_{i,t}$ have opposite signs. This occurs when a target weight element is stationary and the corresponding estimated weight element has already converged around the value of the target element. Subsequent gradient elements start to oscillate in this case and IDBD step-size update rule decreases the corresponding step-size element to reduce the oscillation along

that direction.

Therefore, using a vector step-size parameter and adapting it using the meta descent technique, IDBD increases step-size values so as to track nonstationary weights and decreases it for stationary weights.

Each iteration in IDBD takes $O(n)$ computation where n is the size of the input vector. As such, IDBD meets many of our goals of step-size adaptation algorithms. However, IDBD contains a meta-step-size parameter θ and it is unclear how to set the value of this parameter for a specific problem.

IDBD also requires the user to provide the initial values for the weights and step sizes. The choice of the initial step-size value can also be critical as a very high value can result in the divergence of the algorithm before the adaptation starts to have a strong effect on the values of the step-size parameter. We illustrate these issues later in this thesis.

2.3 The Recursive-Least-Squares (RLS) Filter

A standard supervised learning algorithm is the Recursive-Least-Square (RLS) filter (Ljung, 1998). RLS requires quadratic time and memory computation and can be thought as a matrix step-size adaptation algorithm. The RLS algorithm iteratively minimizes the least-square error—

$$\frac{1}{N}\sum_{t=0}^{N-1}\gamma^{N-t-1}\left[y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right]^2, \tag{2.5}$$

the weighted average of the sample squared error, where $0 < \gamma \leq 1$ is a discounting factor and N is the number of samples.

Given that the number of samples N is large enough so that the matrix $R_N = \sum_{t=0}^{N-1}\gamma^{N-t-1}\boldsymbol{x}_t\boldsymbol{x}_t^\top$ is invertible with probability one, the solution of minimizing Equation 2.5 is—

$$\boldsymbol{w}_N^* = R_n^{-1}\sum_{t=0}^{N-1}\gamma^{N-t-1}y_t\boldsymbol{x}_t.$$

By expanding it in a recursive form and using matrix inversion lemma, the above solution can be written in the following form, which comprise the update rules of the RLS algorithm:

Initialization:

$$\boldsymbol{w}_0 : \text{arbitrary},$$
$$B_0 = \lambda^{-1}\mathbf{I},$$

Updates:

$$\begin{aligned} A_t &= \frac{B_t}{\gamma + \boldsymbol{x}_t^\top B_t \boldsymbol{x}_t}, \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + A_t \delta_t \boldsymbol{x}_t, \\ B_{t+1} &= \gamma^{-1}\left(I - A_t \boldsymbol{x}_t \boldsymbol{x}_t^\top\right) B_t, \end{aligned}$$

where λ is a regularization parameter and $0 < \gamma \leq 1$ is a discounting factor.

By considering the weight update rule, RLS can be interpreted as LMS with a matrix step-size parameter A_t adapted as a function of another matrix B_t.

Having a matrix step-size parameter, RLS require $O(n^2)$ computation, where n is the size of the input vector. In problems where input dimensions is very large, such second order methods can become infeasible. One limitation of RLS is that the update rule for B matrix in RLS does not provide a scope for the elements of the step-size A to reduce over time when its weights are converged on average. This means that estimated weights of RLS would oscillate around the solution in the asymptote.

2.4 K1: Meta Descent for Normalized LMS

The idea of incremental and online adaptation of a vector step-size as in IDBD is transferable to other base learning systems than LMS. Sutton (1992b) proposed K1, which is inspired by RLS, and which resembles and probably more akin to the Normalized LMS. K1 approximates only the diagonal terms of B_t of RLS by a vector $\boldsymbol{b}_t$. It uses a gradient-based rule similar to IDBD in order to have linear-time computation and the adaptive benefits of IDBD. Update rules of K1 in vector

element form is as follows:

$$
\begin{aligned}
\beta_{i,t+1} &= \beta_{i,t} + \theta \delta_t x_{i,t} h_{i,t}, \\
b_{i,t+1} &= e^{\beta_{i,t+1}}, \\
\alpha_{i,t+1} &= \frac{b_{i,t+1}}{\hat{R} + \sum_j b_{j,t+1} x_{j,t}^2}, \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\
h_{i,t+1} &= \left(h_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}\right) \left(1 - \alpha_{i,t+1} x_{i,t}^2\right)^+,
\end{aligned}
$$

where $\theta > 0$ is a meta-step-size parameter and $\hat{R}$ is the estimated observational noise.

Sutton (1992b) empirically demonstrated that K1 outperforms RLS on a non-stationary problem. Computation of each iteration for K1 is linear on the number of inputs.

K1 has an extra parameter $\hat{R}$. Due to its relationship with RLS, it follows from the Kalman filter algorithm (Kalman, 1960; Ljung, 1998) that the value of $\hat{R}$ should ideally be the observational noise variance. It is not clear what value to use in the absence of such knowledge. Sutton (1992b) used a value of one in his experiments and so we do in this study. However, for nonstationary observational noise, the best choice of additional parameter $\hat{R}$ in K1 needs to be adapted over time.

The weight update is normalized at each time-step by a scalar $\hat{R} + \sum_j b_{j,t+1} x_{j,t}^2$, a function of the instantaneous values of inputs. It resembles a standard algorithm—Normalized LMS (NLMS) (Goodwin and Sin, 1984)—where the effective step size is normalized by the instantaneous values of inputs as follows:

$$
w_{i,t+1} = w_{i,t} + \frac{\alpha}{\epsilon + \sum_j x_{j,t}^2} \delta_t x_{i,t},
$$

where ϵ is a small positive constant and α is a fixed, scalar step-size parameter. Due to this similarity, K1 can be thought as parallel to NLMS as IDBD is to LMS.

K1 also has two other parameters to be set: the initial values for β_i and the meta-step-size θ. It is not indicated how to set the value of θ in different problems.

2.5 SMD: Meta Descent for Nonlinear Systems

Schraudolph (1999) proposed an extension to IDBD for nonlinear Systems (also see Schraudolph et al. (2006)). He refers his algorithm as Stochastic Meta Descent (SMD). The derivation of SMD is similar to IDBD except that it does not diagonally approximate the matrix G_t in Equation (2.3).

For general optimization problems, the update rules of SMD in vector forms are as follows:

$$\begin{aligned}
\boldsymbol{\alpha}_{t+1} &= \boldsymbol{\alpha}_t \max\left(\frac{1}{2}, 1 - \theta \boldsymbol{g}_t \circ \boldsymbol{h}_t\right), \\
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \boldsymbol{\alpha}_t \circ \boldsymbol{g}_t, \\
\boldsymbol{h}_{t+1} &= \lambda \boldsymbol{h}_t - \lambda \boldsymbol{\alpha}_{t+1} \circ (R_t \boldsymbol{h}_t) - \boldsymbol{\alpha}_t \circ \boldsymbol{g}_t,
\end{aligned}$$

where $0 \leq \lambda \leq 1$ is a discounting factor, $\boldsymbol{g}_t = \nabla_{\boldsymbol{w}_t} l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ is the gradient of some sample loss function $l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ given sample $\boldsymbol{z}_t$ and $R_t = \nabla^2_{\boldsymbol{w}_t} l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ is the second-order partial-derivative matrix of the sample loss function.

The update rules in the context of linear supervised learning are as follows:

$$\begin{aligned}
\boldsymbol{\alpha}_{t+1} &= \boldsymbol{\alpha}_t \max\left(\frac{1}{2}, 1 + \theta \delta_t \boldsymbol{x}_t \circ \boldsymbol{h}_t\right), \\
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t), \\
\boldsymbol{h}_{t+1} &= \lambda \boldsymbol{h}_t - \lambda \boldsymbol{\alpha}_{t+1} \circ \left(\boldsymbol{x}_t \boldsymbol{x}_t^\top \boldsymbol{h}_t\right) + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t).
\end{aligned} \tag{2.6}$$

The same algorithm in vector-element form:

$$\begin{aligned}
\alpha_{i,t+1} &= \alpha_{i,t} \max\left(\frac{1}{2}, 1 + \theta \delta_t x_{i,t} h_{i,t}\right), \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\
h_{i,t+1} &= \lambda \left(h_{i,t} - \alpha_{i,t+1} x_{i,t} \sum_j h_{j,t} x_{j,t}\right) + \alpha_{i,t+1} \delta_t x_{i,t}.
\end{aligned}$$

Note that, a linear approximation of the exponential increase in step-size is used and a lower-bound is set on the maximum decrease at each time step to guard against negative values.

The update rule of $\boldsymbol{h}_t$ for SMD in Equation (2.6) can be contrasted with that of IDBD in Equation (2.4) to appreciate that the $\boldsymbol{h}_t$ in SMD is not a diagonal approximation of the matrix G_t. The $\boldsymbol{h}_t$ update contains $\boldsymbol{x}_t\boldsymbol{x}_t^\top$, which is $\nabla^2_{\boldsymbol{w}_t}\delta_t^2$, the second-order partial-derivative matrix of the sample squared error. Because of the symmetric nature of this matrix, it can be computed directly in linear time. IDBD diagonally approximates this matrix.

The difference of SMD with IDBD in $\boldsymbol{h}_t$ update can have some significance. It is interesting to know whether keeping the non-diagonal terms of $\boldsymbol{x}_t\boldsymbol{x}_t^\top$ improves the performance of step-size adaptation.

Schraudolph (1999) acknowledged that stability of the algorithm is dependent on the tuning of the meta-step-size parameter θ, indicating that this parameter can be an obstacle in adapting the step-size parameter automatically.

2.6 ALAP: Meta Descent with Normalization

Having a tunable parameter such as the meta-step-size parameter is a major obstacle for meta-descent based adaptation. Although many have acknowledged that the choice of this parameter can be problem dependent (Schraudolph, 1999; George and Powell, 2006), few works have suggested how to set this parameter methodically or heuristically.

Almeida et al. (1998) proposed the only algorithm that we know of deploying a strategy for achieving a problem-independent meta-step-size parameter. Their strategy involves normalizing the meta-descent update at each time step. We call this algorithm the Almeida-Langlois-Amaral-Plakhov algorithm or ALAP. It is proposed for step-size adaptation in general stochastic optimization. Update rules of ALAP for a general optimization problem is as follows:

$$
\begin{aligned}
v_{i,t+1} &= (1-\gamma)v_{i,t} + \gamma(g_{i,t})^2, \\
\alpha_{i,t+1} &= \alpha_{i,t}\left(1 + \theta\frac{g_{i,t}g_{i,t-1}}{v_{i,t+1}}\right), \\
w_{i,t+1} &= w_{i,t} - \alpha_{i,t+1}g_{i,t},
\end{aligned}
$$

where $g_{i,t} = \frac{\partial l(\boldsymbol{z}_t;\boldsymbol{w}_t)}{\partial w_{i,t}}$ is the ith element of the gradient of some sample loss function

$l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ given sample $\boldsymbol{z}_t$, $\theta > 0$ is the meta-step-size parameter and $0 < \gamma \leq 1$ is a discounting factor for the update of the vector elements $v_{i,t}$. The step-size update $g_{i,t}g_{i,t-1}$ is the product of the current and the past sample gradient elements. Therefore, ALAP follows the delta-delta rule for step-size update. The vector elements $v_{i,t}$ are running average of the squared sample gradient elements $(g_{i,t})^2$.

The vector elements $v_{i,t}$ are used for element-wise normalization of the meta-descent update $g_{i,t}g_{i,t-1}$ so that problem dependence of the meta-step-size value is eliminated.

For linear supervised learning, we instantiate the algorithm as follows:

$$
\begin{aligned}
v_{i,t+1} &= (1-\gamma)v_{i,t} + \gamma(\delta_t x_{i,t})^2, \\
\alpha_{i,t+1} &= \alpha_{i,t}\left(1 + \theta\frac{\delta_t x_{i,t}\delta_{t-1}x_{i,t-1}}{v_{i,t+1}}\right), \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1}\delta_t x_{i,t}.
\end{aligned}
$$

ALAP adapts a vector step-size parameter using linear time computation for each iteration. ALAP can be thought as a special case of IDBD where $h_{i,t}$ is $\delta_{t-1}x_{i,t-1}$ from the immediate past, instead of being a trace of it over time. Step-size updates in ALAP can be negative and a lower-bound is necessary to prevent that.

Almeida et al. (1998) reported that $\theta = 0.01$ was appropriate for most of the problems they used in their experiments. ALAP still requires the user to provide an initial step-size value.

2.7 Running-Average-Based Algorithms

Techniques other than the meta-descent strategy have also been developed to adapt the step-size parameter. These algorithms adapt the step-size parameter as a running average of some observable measure. The specific measure depends on the particular algorithm. Most of them adapt only a scalar step-size parameter.

One of the simplest kinds of step-size adaptation algorithms with a running-average based strategy was proposed by Sompolinsky et al. (1995) (also see Barkai

et al. (1995)). We refer to their algorithm as the Sompolinsky-Barkai-Seung (SBS) algorithm. They have proposed their algorithm for binary classification. For a general optimization setting, the update rules of their algorithm are:

$$\alpha_{t+1} = \alpha_t \left(1 + \gamma \left(\theta l(\boldsymbol{z}_t; \boldsymbol{w}_t) - \alpha_t\right)\right),$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha_{t+1}\boldsymbol{g}_t,$$

where $\theta > 0$ is a meta parameter and $0 < \gamma \leq 1$ is a discounting factor.

We instantiate the algorithm for linear supervised learning setting as

$$\alpha_{t+1} = \alpha_t \left(1 + \gamma \left(\theta \delta_t^2 - \alpha_t\right)\right),$$
$$w_{i,t+1} = w_{i,t} + \alpha_{t+1}\delta_t x_{i,t}.$$

The step-size α_t, here, is a running average of past sample squared errors δ_t^2. As SBS adapts only a scalar step-size parameter, this algorithm essentially follows the steepest-descent direction, which might not be appropriate for problems with largely different curvatures in different directions of the error surface and for non-stationary problems with irrelevant inputs. Their algorithm requires user to provide appropriate values for θ, γ and initial step-size value.

Algorithms very similar to SBS are also proposed by others later. For example, Amari (1998) extended the SBS algorithm by expressing the scalar step-size parameter as an exponential function and applying it for a natural gradient-descent rule. Murata et al. (1996) proposed an algorithm where a scalar step-size parameter is adapted as a running average of the L^2 norm of the trace of the past gradients.

An exception of running average-based adaptation is SGD-QN (Stochastic Gradient Descent: Quasi Newton) by Bordes et al. (2009) where a vector step-size parameter is adapted. It is proposed for general optimization. However, when we derived their algorithm for linear supervised learning, the step-size update found to be the same scalar amount for all the elements of the step-size parameter. It essential makes the algorithm a scalar step-size adaptation algorithm.

SGD-QN is proposed for binary classification problems with convex loss functions in linear Support Vector Machine systems. The step-size parameter in their

algorithm diagonally approximates the inverse of the Hessian matrix. The diagonal elements of this matrix are approximated as the running average of the ratio $(w_{i,t+1} - w_{i,t}) \,/\, (g_i(\boldsymbol{z}_t, \boldsymbol{w}_{t+1}) - g_i(\boldsymbol{z}_t, \boldsymbol{w}_t))$, where g_i is $i$the element of sample gradient vector $\boldsymbol{g}$, defined as $\boldsymbol{g}(\boldsymbol{z}_t; \boldsymbol{w}_t) = \nabla_{\boldsymbol{w}_t} l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ and $l(\boldsymbol{z}_t; \boldsymbol{w}_t)$ is a sample loss function, given a sample $\boldsymbol{z}_t$.

The update rules for general optimization are as follows:

$$
\begin{aligned}
w_{i,t+1} &= w_{i,t} - \alpha_t \circ g_i(\boldsymbol{z}_t, \boldsymbol{w}_{t+1}), \\
\alpha_{i,t+1} &= \alpha_{i,t} + \gamma \left(\frac{w_{i,t+1} - w_{i,t}}{g_i(\boldsymbol{z}_t, \boldsymbol{w}_{t+1}) - g_i(\boldsymbol{z}_t, \boldsymbol{w}_t)} - \alpha_{i,t} \right),
\end{aligned}
$$

where $0 < \gamma \leq 1$ is a discounting factor.

For linear supervised learning, we can derive the step-size update rule as in the following:

$$
\begin{aligned}
\alpha_{i,t+1} &= \alpha_{i,t} + \gamma \left(\frac{w_{i,t+1} - w_{i,t}}{g_i(\boldsymbol{z}_t, \boldsymbol{w}_{t+1}) - g_i(\boldsymbol{z}_t, \boldsymbol{w}_t)} - \alpha_{i,t} \right) \\
&= \alpha_{i,t} + \gamma \left(\frac{\alpha_{i,t} \delta_t x_{i,t}}{-\delta_t x_{i,t} + \delta_t x_{i,t} \sum_j \alpha_{j,t} x_{j,t}^2 + \delta_t x_{i,t}} - \alpha_{i,t} \right) \\
&= \alpha_{i,t} + \gamma \left(\frac{\alpha_{i,t}}{\sum_j \alpha_{j,t} x_{j,t}^2} - \alpha_{i,t} \right) \\
&= \alpha_{i,t} \left(1 + \gamma \left(\frac{1}{\sum_j \alpha_{j,t} x_{j,t}^2} - 1 \right) \right).
\end{aligned}
$$

It is easy to note that this step-size update scales each step-size element by the same scalar amount. Therefore, it is a scalar step-size adaptation algorithm in the linear supervised setting.

2.8 Related Theoretical Works

Although a number of step-size adaptation algorithms are developed in the literature, few works have been done on the theoretical guarantees of these algorithms. However, many theoretical works exist that relate to adaptive algorithms and particularly nonstationary problems.

The meta-descent algorithms are based on gradient-descent rules, hence the stability of these algorithms should rely on some conditions of the meta-step-size parameter. However, such conditions of the meta-step-size parameter for the stability of the step-size adaptation algorithms have not been worked out.

Theoretical guarantees on the convergence of stochastic gradient descent without step-size adaptation such as the LMS algorithm already exist. Widrow (1971) provided the following necessary condition on the step-size parameter of LMS for converging on average to the solution:

$$\alpha \leq \frac{2}{\lambda_{max}},$$

where λ_{max} is the maximum eigenvalue of the input correlation matrix $\mathbb{E}\left[\boldsymbol{x}_t \boldsymbol{x}_t^\top\right]$. A stricter bound is as follows (Diniz, 2002; Widrow and Stearns, 1985):

$$\alpha \leq \frac{2}{\mathbb{E}\left[\boldsymbol{x}_t^\top \boldsymbol{x}_t\right]}.$$

Widrow et al. (1976) derived an expression for *misadjustment*, the ratio between average excess MSE and the minimum MSE, for LMS in a nonstationary setting. They assumed that the target weights are Markovian, drifting by adding white noise of equal variance for each element, and that the inputs are stationary. Under these assumptions, they showed that the total misadjustment is a sum of the misadjustment due to gradient noise and the misadjustment due to lag in tracking target weights. The former is proportional to the step-size value whereas the latter is inversely proportional to it. Therefore, an optimal value of the step-size parameter in LMS exists which minimizes the total misadjustment error.

Theoretical works on a general setting that covers both stationary and nonstationary problems can also be found in regret minimization literature. For example, Zinkevich (2003) introduced a gradient-descent algorithm for online convex programming problems and established regret bounds for the algorithm on problems with both stationary and nonstationary target weights.

2.9 Discussion

Among the existing algorithms, only the meta-descent ones adapt a vector step-size parameter. All the running-average-based algorithms adapt only a scalar step-size parameter. As such, meta-descent algorithms are closer to achieving our goals of step-size adaptation. We focus more on the meta-descent algorithms in the following chapters and use running-average algorithms only for comparison.

All these step-size adaptation algorithms contain one or more tunable parameters in them. Most of the works introducing these algorithms demonstrate their results using only one value of these parameters and some mention to manually tune them to obtain the best values. Only ALAP considered this problem methodically and deployed a normalization technique in the meta-descent update in order to obtain a problem-independent meta-step-size parameter. It is interesting to know if the normalization technique is effective on widely different problems.

Chapter 3

Tests on Toy Problems

We present our first contribution in this chapter. We empirically evaluate a number of step-size adaptation algorithms on toy test problems to determine whether these algorithms can adapt their step-size parameters effectively. We also investigate how sensitive these algorithms are to their tunable parameters. For the first time we empirically demonstrate that the step-size adaptation algorithms cannot be effective without tuning of their meta parameters.

Nonstationary problems with a number of irrelevant inputs are appropriate for appreciating the advantage of a vector step-size adaptation algorithm. Use of a vector step-size parameter can also provide substantial advantage in problems with different input variances, which create different curvatures in the performance surface in different directions. However, a nonstationary problem is a more challenging case for automatic vector step-size adaptation as it requires continuous adaptation of the step-size parameter for tracking the solution. Hence, we focus only on nonstationary problems for evaluating adaptation of a vector step-size parameter and do not consider problems with different variances among the inputs.

It is interesting to know whether the meta-descent algorithms, which use a vector step-size parameter, can effectively adapt it in nonstationary problems. The effectiveness of these algorithms can be verified by comparing their performance with LMS, which use a fixed, scalar step-size parameter. For comparison, we use IDBD, K1, SMD, ALAP and SBS, which are already described in Chapter 2. Time and memory complexity of all these algorithms is linear, which is the case of our interest. All of these except SBS are meta-descent algorithms. SBS is a running-

average-based algorithm and adapts only a scalar step-size parameter.

We also investigate whether these algorithms can adapt the step-size parameter in a problem-independent manner. All these algorithms contain one or more tunable parameters. A step-size adaptation algorithm that is sensitive to the choice of its tunable parameters is less interesting with respect to our goals of step-size adaptation, because an adaptation algorithm is needed to get rid of the tuning of a parameter, i.e, the step-size parameter, in the first place. It is important to know whether the existing step-size adaptation algorithms can adapt effectively without requiring any manual tuning across problems.

A good way of investigating the problem dependence of the choice of the tunable parameters is by experimenting the algorithms on different problems. We use an idealized toy problem from the work by Sutton (1992b) and generate more toy problems by varying some of the problem's statistics. These problems are supervised regression learning problems, and contain nonstationarity, and irrelevant inputs. All the algorithms experimented here predict their target output as a linear combination of the inputs.

We explore two issues of the algorithms with their tunable parameters: robustness and sensitivity. We find whether the algorithms perform robustly by adapting the step-size parameter using the same setting of the tunable parameters in all the toy problems. We also investigate whether performance of the algorithms is sensitive to the choice of the parameters within the same test problem.

To investigate robustness and sensitivity of these algorithms, we compare the performance of each algorithm across different toy problems. We vary two different tunable parameters of these algorithms at a time: the meta parameter and the initial step-size value. For the meta-descent algorithms, the meta parameter is the meta-step-size parameter. We find whether an algorithm can adapt the step-size parameter effectively, i.e., perform significantly better than LMS on all the toy problems for the same choice of their tunable parameters.

3.1 Description of Problems and Setup

Here we describe three toy test problems that we use to empirically investigate the existing step-size adaptation algorithms. Problem 1 is an idealized toy test problem taken from Sutton (1992b). Problem 2 and 3 are variations of problem 1.

In problem 1, the data set contains 30,000 sequential data samples. Each sample contains a 20-element vector of inputs and a scalar target output. Inputs were independently drawn from a normal distribution with zero mean and unit variance. Outputs were calculated as a linear combination of the inputs. An observational, white noise with zero mean and unit variance was added to the linear combination. All the target weights were initialized to zero and 15 of them always remain so. Therefore, only five of the inputs are relevant to the output. For the five target weights that correspond to the relevant inputs, a weight drift, drawn independently from a zero mean, unit variance normal distribution, was added at each time step. Therefore, these five target weights are nonstationary, forming a simple random walk. Variances of inputs and weight drifts are two statistics of the problem and a class of different problems can be generated by choosing different values for these variances.

Problem 2 differs from problem 1 in that the variance of the weight drifts corresponding to the five relevant inputs is 100, instead of 1. As the same randomization sequence was used for both problems, the magnitudes of the target weights for the second problem are ten times larger than the first one. Everything else between problem 1 and 2 remains unchanged. Therefore, problem 2 is also a nonstationary problem having similar properties as problem 1 except that the variance of the target output is 100 times larger than problem 1. Using the same algorithm on both problems, it can be expected that Mean Squared Error (MSE) in problem 2 will also be 100 times larger than problem 1.

Problem 3 differs from problem 1 only by the scale of the variance of inputs. In problem 3, all the inputs have a variance of 100. Therefore, the magnitudes of the inputs for problem 3 are ten times larger than problem 1. Problem 3 still remains a nonstationary problem, however 100 times larger input variance results in 100

times larger target-output variance.

In all the experiments in this chapter, each algorithm observed a data sample at each time step and predicts the value of the current target output as a linear combination of the current inputs. Every run was repeated 30 times using different randomization and the average performance is shown. For both ALAP and SBS γ was set to 0.01. For SMD, λ was set to 1.

3.2 Comparison of Performance

We compared performance of IDBD, K1, SMD, ALAP, SBS, RLS and LMS on problem 1. Sutton (1992b) demonstrated performance of IDBD, K1, RLS and LMS on this problem using the same setup. We reproduced the result here to verify the findings by Sutton (1992b) and find the performance of SMD, ALAP and SBS, in addition.

We varied the meta-step-size parameter θ for IDBD, K1, SMD, and ALAP, the meta parameter θ for SBS, the step-size parameter α for LMS and $1-\gamma$ for RLS in this experiment. The step-size parameter α for LMS and $1-\gamma$ for RLS were varied between 10^{-3} to 1 by taking 10 values, equally spaced in logarithmic scale. For the rest of the algorithms, their corresponding parameters were varied between 10^{-9} to 10^{2} by taking 17 values, equally spaced in logarithmic scale. For each choice of these parameter values, RMSE over the last 10,000 time steps was averaged over 30 runs. For all these algorithms except K1, initial step-size, be it scalar or vector, is $0.1/$ (# of inputs) $= 0.005$. For K1, the initial value of the elements of $\boldsymbol{\beta}$ vector was chosen to be $\log(0.1)$.

Our particular experimental methodology relies on comparing performance of different algorithms for different parameter values. Although many of the algorithms have more than one parameters, we vary only one of them, the meta parameter in most cases, to compare performance in each experiment. Whenever we state that one algorithm performs better than the other on a particular problem, we imply that, for at least one of the parameter settings, performance of the former algorithm is better than the performance for all of the parameter settings of the latter

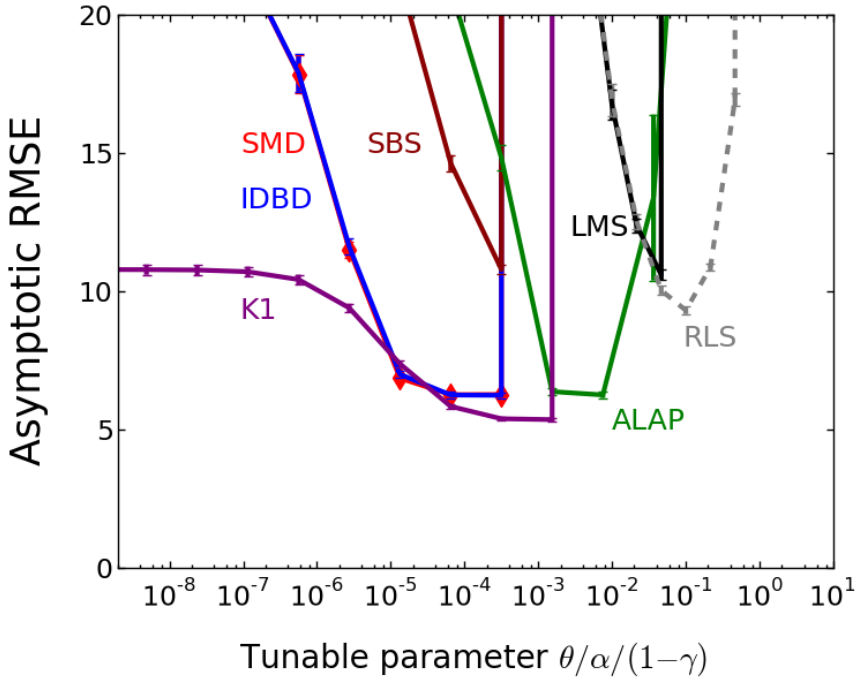

Figure 3.1: Performance of several existing step-size adaptation algorithms: IDBD, K1, ALAP, SMD and SBS for different values of their meta parameters θ shown on a toy nonstationary problem. Performance of LMS for different step-size values α and RLS for different values of $1-\gamma$ are also shown. The meta-descent algorithms IDBD, K1, ALAP and SMD perform better than LMS.

algorithm. Although we demonstrate results for varying only one of the parameters in such cases, the statement also holds for the variations in the other parameters, too. A possible weakness of such comparison is that an algorithm which performs better than another algorithm for a particular setting of its parameters can be so sensitive to the choice of its parameters that it requires substantial fine tuning. Cases where performance is more sensitive to its parameter choice than other algorithms, superior performance of the former one does not lead to a straightforward advantage. But, we demonstrate the performance curve for all of the parameter values that we choose and hence the sensitivity of each algorithm to its parameter is readily demonstrated. We notify such sensitivity of algorithms each time they occur. Therefore, our methodology is not prone to the stated weakness.

Meta-descent algorithms: IDBD, K1, SMD and ALAP performed better than LMS, as shown in Figure 3.1. Best average RMSE of LMS is slightly above 10 for a narrow range of step-size values between 10^{-2} to 10^{-1}. As the meta-descent

algorithms can be thought as adapting the step-size parameter of LMS in a vector form, it can be concluded that they are adapting the step-size parameter effectively in this nonstationary problem. These results are in harmony with those by Sutton (1992b) except that the previous work has shown results only for one run and the parameter values were varied in linear scale.

Meta-descent algorithms perform better than LMS on this problem because these algorithms adapt the elements of the vector step size appropriately. For the relevant inputs, they adapt the corresponding step size to high values. On the other hand, for irrelevant inputs, the corresponding step-size values are drawn close to zero.

SBS did not achieve a better performance than LMS for any values of θ. Its best performance is also achieved for a narrow range of θ values. Best performance of RLS is not lower than 10 and hence is outperformed by meta-descent algorithms.

Performance of IDBD, shown in blue, and performance of SMD in red with diamond-shaped markers were almost similar. Therefore, no advantage of the extra terms added to the $\boldsymbol{h}_t$ update of SMD can be seen in this problem. Such was expected, as the update rules of SMD are almost the same as IDBD. We find similar results between IDBD and SMD for all the experiments throughout this thesis. Hence, we only show the results for IDBD, in the rest.

3.3 Robustness and Sensitivity

We compare robustness and sensitivity of IDBD, K1, ALAP and SBS on the choice of their meta parameters and initial step-size parameter across problem 1, 2, and 3. These problems differ in the variance of target output by orders of magnitude. Therefore, MSE performance of these algorithms, shown in these plots, were normalized by the variance of the target outputs, estimated over last 10,000 time steps.

We define robustness as an algorithm's ability to get effective performance across different problems with the same setting of its parameter values. By sensitivity, we refer to the range of parameter values for which an algorithm achieves its best performance within the same problem. If an algorithm obtains its best per-

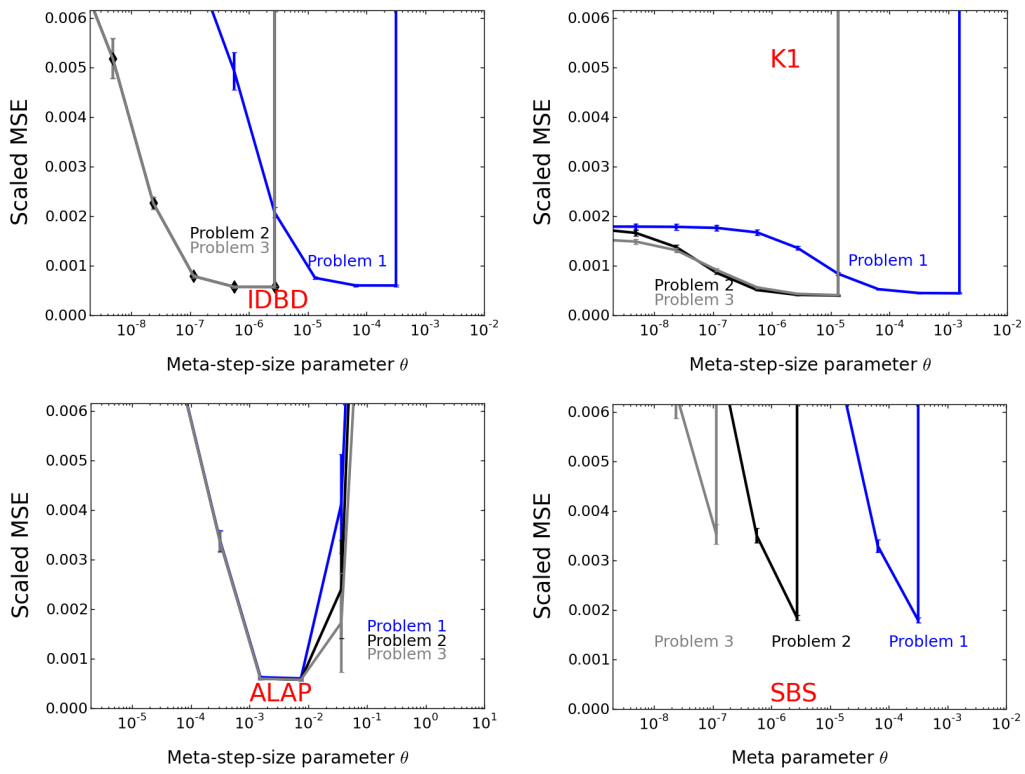

Figure 3.2: Performance of IDBD, K1, ALAP and SBS for different meta parameters on problem 1, 2, and 3, demonstrating robustness and sensitivity. All the algorithms except ALAP need different meta-parameter values for achieving their best performance on all problems.

formance for a narrow range of parameter values, we consider it to be too sensitive to the choice of its tunable parameter.

For problem 1, 2, and 3, Figure 3.2 shows performance curves of IDBD, K1, ALAP and SBS, separately. These results discover that IDBD, K1 and SBS are not robust algorithms in that they require tuning of their meta parameter to different values for different problems. For example, IDBD achieved its best performance for a range of θ around 10^{-4} on problem 1. Problem 2 and 3 created 100 times larger MSE for IDBD than problem 1 and the best choice of θ shifted to 100 times lower value, around 10^{-6}. There is no overlap between these two ranges. Similar is observed for K1 and SBS.

ALAP on the other hand demonstrates robust performance on three of the problems. ALAP performed best for a range of meta-step-size values between 10^{-3} to 10^{-2} on all these problems. It indicates that the normalization in meta-descent

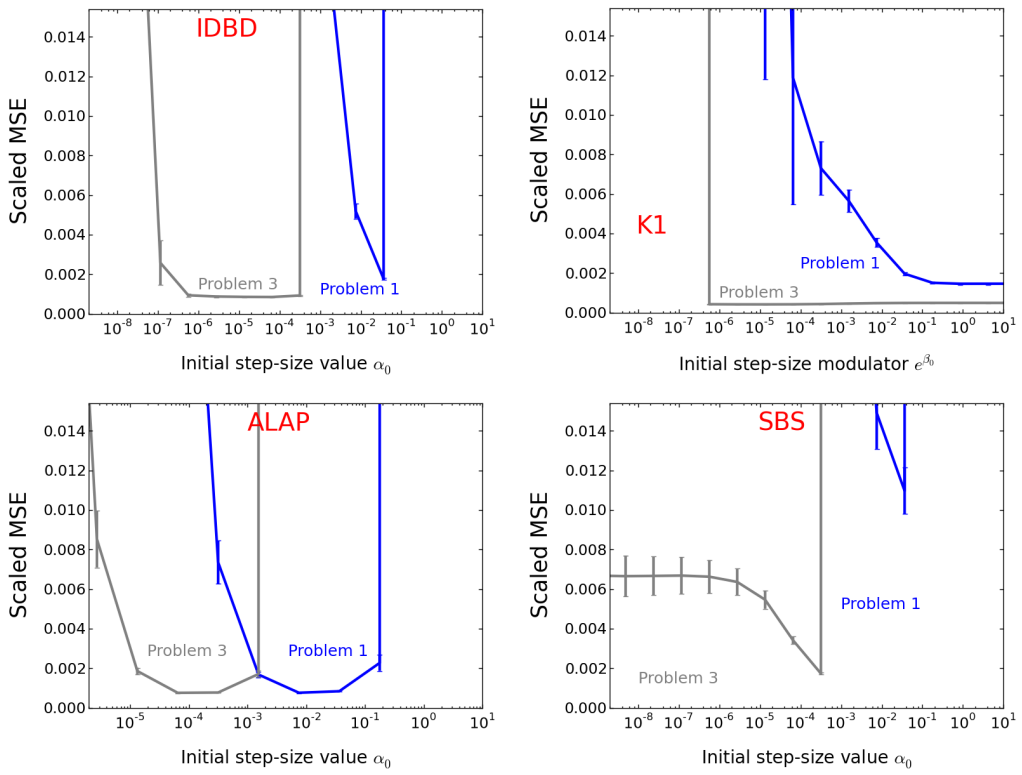

Figure 3.3: Performance of IDBD, K1, ALAP and SBS for different initial step-size values on problem 1, and 3. All the algorithms except K1 need different initial step-size values for achieving their best performance.

update deployed by ALAP was effective in these toy test problems.

SBS is the most sensitive to the choice of the meta parameter among all the algorithms. It achieved its best performance for a narrow range of θ.

All these algorithms are sensitive to some extent in that performance largely varies on the same problem depending on the value of the meta parameter. On a single problem, performance of K1 is less sensitive. Its curves are relatively flatter than other algorithms.

Figure 3.3 shows performance of IDBD, K1, ALAP and SBS separately on problem 1, and 3 for different initial step-size values. Performance of IDBD, ALAP and SBS was not robust on their initial step-size values. There is no overlap between the best values of initial step size for these algorithms.

For K1, the initial condition for step-size parameter was modulated by setting initial values for $e^{\boldsymbol{\beta}_t}$. Results show that performance of K1 was more robust and less sensitive to initial conditions of step-size parameter. A value larger than 10^{-1}

can be chosen to achieve its best performance in both problems.

Sensitivity of K1 on the initial step-size condition is less than other algorithms within a single problem. The range of best values is flat for a large region.

SBS, on the other hand, is the most sensitive to the initial step-size values. Its best performance can be achieved for a narrow range of initial step-size values.

3.4 Discussion

In this chapter we have established that step-size adaptation algorithms can effectively adapt the step-size parameter on nonstationary problems. Meta-descent algorithms—IDBD, SMD, K1 and ALAP—achieved at least two times better performance than LMS. SBS, which is a running-average-based algorithm adapting a scalar step-size parameter, cannot achieve a performance better than LMS. SBS has been demonstrated by Barkai et al. (1995) to be effective on nonstationary problems. However, it was demonstrated on problems with few inputs. In the problems that we tested here, there are 20 inputs, where 15 of them are irrelevant. Advantage of SBS in tracking nonstationary target is diminished due to stationary target weights corresponding to irrelevant inputs, for which the scalar step-size parameter creates large oscillation noise.

Our results indicate that most of the step-size adaptation algorithms require problem dependent parameter tuning for effective adaptation. Best values of meta parameters were different in these three problems for all the algorithms exept ALAP. Therefore, most of these algorithms cannot adapt the step-size parameter in a completely automatic manner.

The normalization in meta-descent update deployed by ALAP was effective in these problems. Almeida et al. (1998) found $\theta = 0.01$ to be appropriate for the problems they have experimented with ALAP. This value can also achieve best performance on the three nonstationary problems that we have tried. It indicates that the normalization strategy can be useful in achieving a step-size adaptation algorithm that does not require parameter tuning across problems.

We also found that Initial step-size value requires problem-dependent tuning for

most of these algorithms. It can be surprising as these algorithms are adapting the step-size values over time. However, it takes a number of time steps before the step-size value is drawn near the correct value. If a large initial step-size value is chosen, the algorithm becomes unstable before the adaptation procedure sets the step-size parameter properly.

Choosing an initial step-size value small can also affect performance, as shown by our results. It is also surprising because we demonstrate performance in MSE that is calculated over the last 10,000 times steps among total 30,000 time steps. If an effective step-size adaptation algorithm can adapt the step-size parameter to appropriate values within the first 20,000 time steps, then the affect of a low initial step-size value should not remain in the next 10,000 time steps. Therefore, these algorithms cannot adapt step-size parameter from a very low initial value to appropriate values within this time range. However, effect of a very low initial step-size value should eventually be diminished if the experiments are run for a longer time.

K1 with a high initial value for $e^{\boldsymbol{\beta}_t}$, e.g., as high as 10^{-1} or larger is effective across the problems we experimented. It is probably due to the fact that the effective step-size value of K1 is a normalized term, being modified at every time steps depending on the instantaneous values of the inputs. We state the weight update rule for K1 again here:

$$\alpha_{i,t+1} = \frac{b_{i,t+1}}{\hat{R} + \sum_j b_{j,t+1} x_{j,t}^2},$$
$$w_{i,t+1} = w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t},$$

where $b_{i,t+1} = e^{\beta_{i,t+1}}$ and $\beta_{i,t}$ is adapted using a gradient-based rule. This normalization helps K1 to reduce the impact of a large initial $e^{\beta_{i,t}}$ on the effective step-size value soon. When $e^{\boldsymbol{\beta}_t}$ is chosen high, the effective step-size value $\alpha_{i,t+1}$ is more likely to get reduced to a small value. On the other hand, for a small initial value of $e^{\beta_{i,t}}$, $\alpha_{i,t+1}$ can be small and it needs time for the adaptation procedure to adapt $\beta_{i,t}$, and hence the effective step size, to an appropriate high value.

The idea of normalization in the meta-descent update seems promising in making a step-size adaptation algorithm less sensitive to its meta-step-size parameter.

Chapter 4

Tests on Robot Data

In this chapter we use real-world, robot data for evaluating performance of step-size adaptation algorithms.

We use a large data set from a sensor-rich robot called Critterbot and experiment with IDBD, K1 and ALAP on this data. This data set consists of high-dimensional sensor signals collected by running the Critterbot for a long time. The tasks for the algorithms are to predict six signals of the robot in the next time step by using all the signals from current the time step. Predicting such signal values by establishing a linear relationship with all the signals from previous time step is a complex task. As the robot is run for a long time and faces changes in surrounding environment while it moves, the relationship between signals may also change over time. Therefore, such problems are potentially nonstationary. Experimenting the algorithms on such data set necessarily pose them on a large, real-world problem.

In all of our robot experiments, SMD performed similarly to IDBD. SBS also did not perform better than LMS in any problem. These results are similar to those in Chapter 3. In that chapter we have already found that performance of SMD is almost similar to IDBD. On the other hand, SBS adapts only a scalar step-size parameter and was not found performing better than LMS on any of the toy non-stationary problems. Hence we do not demonstrate the results of SMD and SBS in this chapter.

We will demonstrate, in this chapter, that all the algorithms are sensitive to the meta-step-size parameter by orders of magnitude. We will also find that ALAP, which performed robustly on toy test problems, has a different best meta-step-size

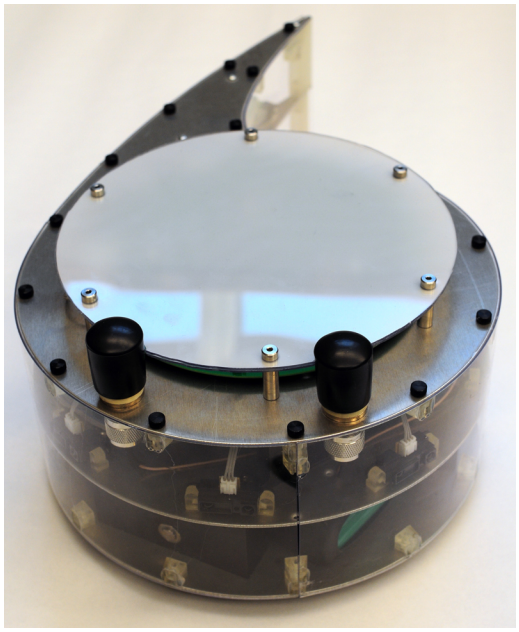

Figure 4.1: The Critterbot from the front. The tail of the robot can be seen at its backside. It has three omni-directional, motor-driven wheels at the bottom. A number of sensor fields surround the Crittebot's body.

value on robot problems.

4.1 Description of Critterbot

Critterbot, a custom built mobile robot (RLAI, 2010), is used to generate the data set. Critterbot is a sensor-rich robot with a comma-shaped frame where a 'tail' facilitates object interaction. Picture of the robot is shown in Figure 4.1. It is driven by three omni-directional, motor-driven wheels positioned at 120 degrees of separation. Critterbot contains a diverse set of sensors, which include sensors for sensing the environment such as ambient light, infrared light, heat and magnetic fields as well as sensors capturing proprioceptive information such as motor speed, rotational velocity, wheel acceleration, motor current, motor voltage, motor temperature, battery voltage, etc.

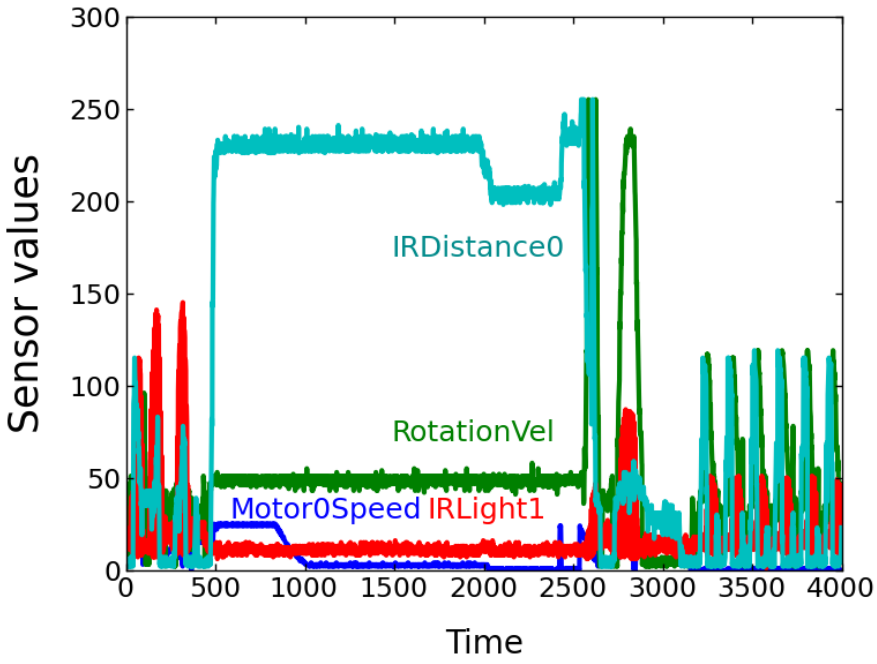

Figure 4.2: Values of sensors—*Motor0Speed*, *RotationVel*, *IRLight1* and *IRDistance0* over 4000 time steps.

4.2 Description of Problems and Setup

Critterbot was run continuously for $\sim$ 40 hours using a random policy inside a closed testing environment and the sensor data were sampled at a frequency of $10ms$. Our data set contains first 3 million samples from this data, which is $\sim$ 8 hours and 20 minutes of Critterbot activity. Sensor data contains time record of 56 sensors. 6 sensors among them are chosen for predicting their values in the next time step. They are listed here: *Light2*—signal from one of the light sensors, *Motor0Speed*—speed of one of the motors, *Motor0Current*—current of one of the motors, *AccelZ*—acceleration of one of the wheels, *RotationVel*—rotational velocity and *IRDistance0*—one of the infrared distance sensors sensing distance from a nearby obstacle. Values of all these signals are presented using integers in specific ranges, which are different between the signals.

These sets of prediction problems are challenging for the step-size adaptation algorithms as the scales of the values of the six target signals are largely different. For example, values of the IRDistance0 sensor varied between 1.0 to 255 over

the 3 million samples, with a variance of about 60. On the other hand, minimum, maximum and variance of RotationVel were -454.0, 503.0 and 157.26, respectively. This may lead to different best meta-step-size values in predicting different sensors. Figure 4.2 shows the time series of the sensor values for *Motor0Speed*, *RotationVel*, *IRLight1* and *IRDistance0* over 4000 time steps, which is equivalent to 40 seconds period of Critterbot activity. As can be seen from the plot, characteristics of the signal values are changing over the time. Correlations also exist between the signals, which means one signal can be used to predict another signal. Predicting such a sensor value at next time step as a linear combination of all the sensors from current time step is a highly nonstationary task and step-size adaptation can play an important role here.

As we only focus on testing the advantage of adapting a vector step-size parameter in nonstationary environment, we do not focus the other case where such adaptation can also have an advantage, i.e., problems with different curvatures in different directions of the performance surface. Such problems typically arise when variance of inputs are different. To focus only on the challenge of nonstationarity, we normalize all of our inputs by subtracting the sample mean and dividing by the sample standard deviation. However, all the experiments can also be repeated using the raw values for the inputs to make the problem more complex and difficult. In the experiments of this thesis, we only focus on inputs with similar variance and hence use the data with normalized inputs.

The data set with 3 million samples is divided into 30 sets so that each one is 100,000 time-steps long. All the inputs in each set are normalized by subtracting the sample mean of the input values for the corresponding set and dividing the result by the sample standard deviation of the particular set. A bias input, which is always one, is added to the set of inputs. Therefore, there are 57 inputs linearly predicting a target signal in the next time step.

For the prediction of each sensors, an algorithm is run on these 30 sets separately and the performance is averaged. Performance is measured in MSE, time-averaged over last 50,000 time-steps for each data set and then averaged over 30 data sets. Then the average MSE is divided by the squared difference between the overall

minimum and maximum values of the signals.

Estimated weights w_i for all the algorithms are initialized to zero. We find the initial value of step-size of IDBD for which the algorithm does not diverge, after several trial and error. This value is $\frac{0.0001}{\text{\# of inputs}} = \frac{0.0001}{57} = 1.754 \times 10^{-06}$ for each elements. Same is used for ALAP. For ALAP μ is set to 0.01. Initial $e^{\boldsymbol{\beta}_i}$ values for K1 is set to 0.1.

4.3 Results

In this section, we compare some of the meta-descent algorithms and find their robustness and sensitivity on their meta-step-size parameter by experimenting them on Critterbot data.

IDBD, K1, ALAP and LMS were run on the tasks of predicting six different sensor values of Critterbot in the next time step. Each algorithm was run for a range of values for their meta-step-size parameter (or the step-size parameter for LMS). For a choice of the parameter, each algorithm is run on 30 data sets and the performance in scaled MSE over last 50,000 time steps were averaged over 30 sets. This way, we can find for what value of the parameters these algorithms perform the best. We can compare the best performance among the algorithms. We find how the choice of the best value of the meta-step-size parameter vary within (sensitivity) and across problems (robustness).

IDBD performed better than LMS in all the problems, as shown in Figure 4.3. For example, in the IRDistance0 prediction problem, IDBD with the meta-step-size value of 10^{-6} achieves its best performance of a value about 0.000015 in scaled MSE, which is about four times better than the best performance of LMS.

ALAP did not achieve a performance better than LMS in any of these problems. The best performance of LMS is about two times better than ALAP in most of the problems.

K1 as well as IDBD has shown the best performance in all these six problems. Performance of K1 was almost the same for every value of meta-step-size parameter. However, the value where K1 starts to diverge is drastically different for differ-

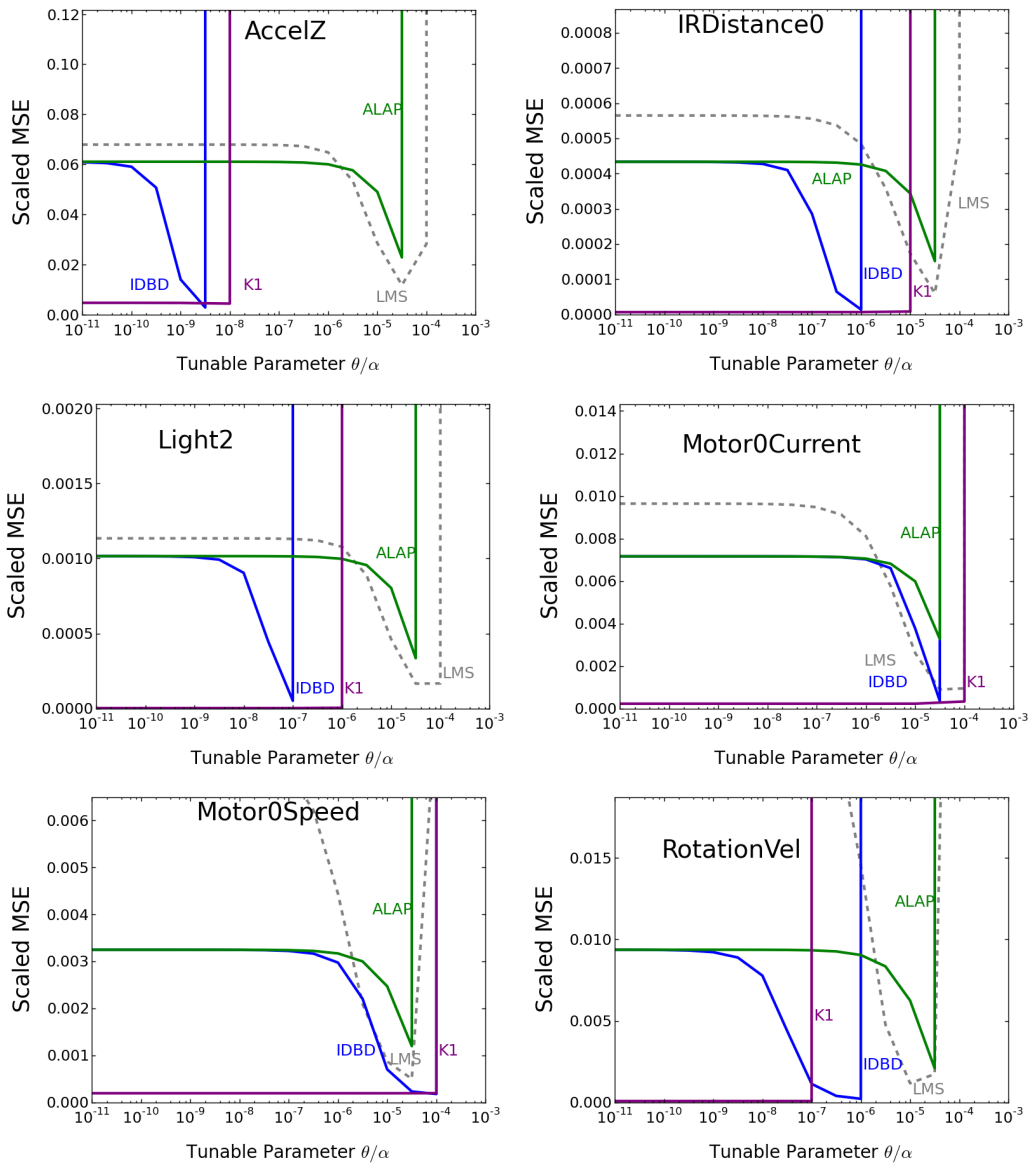

Figure 4.3: Performance of IDBD, K1 and ALAP for different meta parameters on the problems of predicting six sensor values of Critterbot in the next time step. Performance of LMS for different fixed step-size values is also shown.

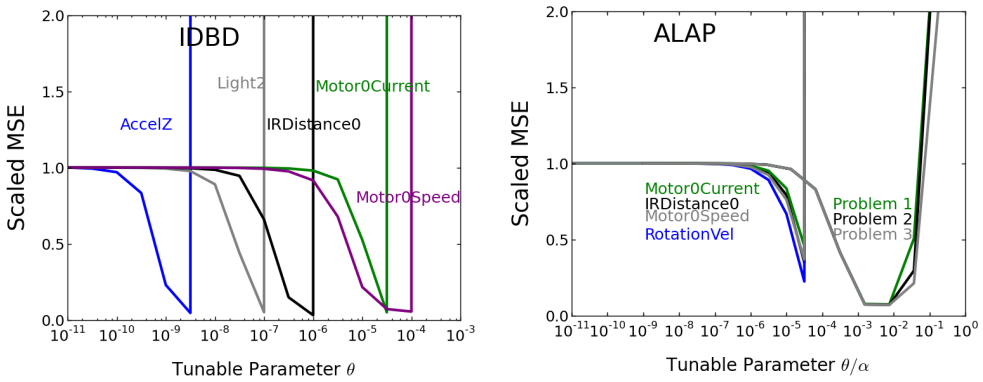

Figure 4.4: Performance of IDBD and ALAP demonstrating the shifts in the best choice of meta-step-size parameter. Performance is scaled so that the values for the smallest meta-step-size parameter (10^{-11}) for different problems are the same. For ALAP, results on toy problems are also shown.

ent problems. For example, in the problem of predicting AccelZ signals, K1 starts to diverge for a meta-step-size value more than 10^{-8}. But, for the Motor0Current prediction problem, this value is above 10^{-4}.

The best value of the meta-step-size parameter for IDBD is largely different between the problems. Left panel in Figure 4.4 shows performance of IDBD for five different problems. Performance values in MSE are scaled so that the performance for the smallest meta-step-size value (10^{-11} here) is always 1. The best value for the problem of predicting AccelZ signals is between 10^{-9} to 10^{-8} and for Motor0Speed, it is 10^{-4}.

Although, ALAP performed poorly on the Critterbot data, its best value is achieved for the same range of meta-step-size values. Right panel of Figure 4.4 shows performance of ALAP for four prediction problems on Critterbot data. Its best performance is achieved for θ between 10^{-5} and 10^{-4} for each case. Performance curve of ALAP on three toy test problems from Chapter 3 is also given in this figure for comparison. It is clear from this figure that the range of best values of meta-step-size parameter has shifted from 10^{-3} to 10^{-2} for toy test problems to values between 10^{-5} and 10^{-4} on the Critterbot data. It clearly establishes that ALAP is not completely robust across different classes of problems.

4.4 Discussion

Some of the results from the toy test problems are reinforced in the results of this chapter. We find that step-size adaptation can perform better than LMS in large, complex problems. None of the algorithms, however, are found to be robust on the meta-step-size parameter.

IDBD is sensitive to the choice of its meta-step-size values. Best performance is achieved for a narrow range of values. Performance is poor, i.e., MSE is high, when a meta-step-size value lower than the best value is chosen. A higher value than the best meta-step-size value results in divergence. It indicates that, the meta-step-size parameter should be tuned very carefully into a small range of values in order to receive the benefit of step-size adaptation from IDBD. It also requires to tune the value again when the problem is different.

K1 performs the best in all the problems that we experimented here, but the meta-step-size value for which it starts to diverge is different for different problems. K1 has a different base setting than LMS. Its effective step-size value is normalized by a function of the power of the inputs at every time step. Superiority of the performance of K1 is not only due to its step-size adaptation strategy but also due to this normalized LMS (NLMS) setting. It is best to consider algorithms such as IDBD or ALAP that directly adapt the step-size parameter of LMS to be candidate step-size adaptation solutions for LMS and consider K1 to be a step-size adaptation solution for NLMS. As K1 diverges for different values in different problems, it demonstrates that K1 also needs problem-dependent tuning of its meta-step-size parameter.

We find that ALAP is not independent of the tuning of its meta-step-size parameter. Almeida et al. (1998) mentioned the value of 10^{-2} to be working in most of their experiments. That value also achieved the best performance in the three toy test problems that we demonstrated in Chapter 3. But, ALAP diverges in all the problems on Critterbot data for this value of meta-step-size parameter. The best value now has shifted between 10^{-5} and 10^{-4}. Certainly, ALAP is more robust on the choice of its meta-step-size parameter compared to IDBD, indicating that the

normalization strategy helps to achieve a problem independent meta-step-size parameter to some extent. However, it does not achieve the goal of being completely automatic and its performance has also considerably degraded, i.e., become worse than LMS, in Critterbot prediction problems.

A step-size adaptation algorithm that can adapt the step-size parameter effectively and in a completely automatic manner is still absent. In the following chapters, we work toward achieving such an algorithm.

Chapter 5

Meta-Normalization: Normalization in Meta Descent

This is the first chapter where we introduce new algorithmic ideas. We provide some insights about why IDBD needs to be tuned to different meta-step-size values on different problems. We combine new techniques with IDBD in order to make the resulting algorithm less sensitive to the meta-step-size parameter. These techniques revolve around the idea of normalizing the meta-descent update term. This idea is similar to that used in ALAP and we refer to it as *meta-normalization* technique.

Although, this idea can be used both in combination with IDBD and K1, we demonstrate experimental results only for the IDBD versions. Our preliminary experiments with the K1 versions indicate similar results to the IDBD ones. We leave the comprehensive experimentation of the K1 counterparts for future works.

5.1 New Meta-Normalization Algorithms

The best choice of the meta-step-size parameter for IDBD varies across problems and there is no straightforward way to set the value of this parameter without manually tuning it. To find some insights about how different statistics of a problem interplay with the choice of the meta-step-size parameter, we recall the results of IDBD from the toy test experiments in Chapter 3. Three nonstationary toy problems were used. In problem 2, variance of weight drift was 100 times more than problem 1. In problem 3, variance of the inputs was 100 times more than that of problem 1. We reproduce the results of IDBD on these three problems again in Figure 5.1.

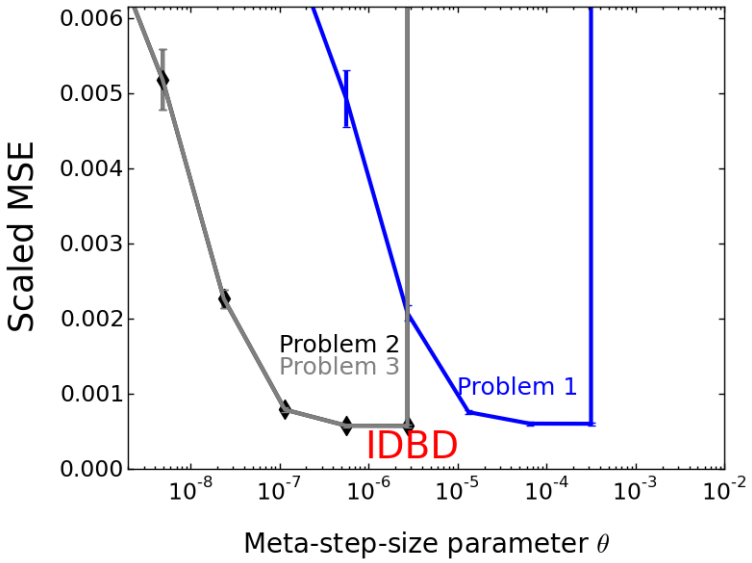

Figure 5.1: Performance curve in scaled MSE for IDBD on the non-staionary, toy-test problems. The range of the best meta-step-size values shifts to lower values from problem 1 to problem 2, and 3.

The curve for problem 1 shifted to lower meta-step-size values in problem 2 and 3. The MSE also shifted to higher values for problem 2 and 3, however it is not apparent from this figure as the performance is shown in scaled MSE for convenience of comparison. MSE was divided by the variance of the target output and it was 100 times larger for problem 2 and 3 than problem 1. Interestingly, the best range of meta-step-size values has also shifted to 100 times lower values from problem 1 to problem 2 and 3.

It is apparent that choosing a suitable meta-step-size value may depend on the variance of the target weights or the variance of the inputs or for both cases, on the variance of the target output. If the meta-step-size parameter is normalized by the variance of the target output, the same range of values for this parameter would perhaps apply to all three problems in a similar way. When the variance of the target output is not known a priori, a running estimate can be used.

The update rules of IDBD reveals some more insights about the interaction of the choice of meta-step-size value with different observable measures of the problem. To have some idea on how the meta-step-size affects the step-size in IDBD, we relate them in the following way:

$$
\begin{aligned}
\alpha_{i,t+1} &= e^{\beta_{i,t+1}} \\
&= e^{(\beta_{i,t}+\theta\delta_t x_{i,t} h_{i,t})} \\
&= e^{\log \alpha_{i,t}} e^{\theta\delta_t x_{i,t} h_{i,t}} \\
\therefore \alpha_{i,t+1} &= \alpha_{i,t} e^{\theta\delta_t x_{i,t} h_{i,t}} \qquad (5.1)
\end{aligned}
$$

From the equation (5.1), we find that $e^{\theta\delta_t x_{i,t} h_{i,t}}$ is a scaling factor for the step-size $\alpha_{i,t}$. The unit of $\alpha_{i,t+1}$ and $\alpha_{i,t}$ is the same, as they both are step-size values for two successive time steps. Therefore, the scaling factor $e^{\theta\delta_t x_{i,t} h_{i,t}}$ and hence the step-size-update term $\theta\delta_t x_{i,t} h_{i,t}$ should be free of units, i.e., the scale of the magnitude of $\theta\delta_t x_{i,t} h_{i,t}$ should not be affected for similar performance when the scale of a statistic such as the variance of target output is different between problems.

We further verify our claim that the scale of the magnitude of step-size update $\theta\delta_t x_{i,t} h_{i,t}$ should remain the same even if the variance of the target output is different between two problems by observing the traces of $\theta\delta_t x_{i,t} h_{i,t}$ over time for the three toy problems. As the best range of meta-step-size values has shifted to 100 times lower values from problem 1 to problem 2 and 3, the meta-step-size value of 10^{-4} for problem 1 is similar to the meta-step-size value of 10^{-6} for problem 2 and 3. We plot the trace of $\theta\delta_t x_{i,t} h_{i,t}$ for $i = 0$ by using the meta-step-size parameter $\theta = 10^{-4}$ for problem 1 and $\theta = 10^{-6}$ for problem 2 and 3 in the top panel Figure 5.2. As expected, the values are exactly the same and completely overlap. We have shifted the curves in time so that all of them can be visible. We have shown it only for one of the elements with $i = 0$, however the same is observed for all of the elements.

Then why does the choice of the meta-step-size parameter depend on the change in the scale of the target output variance between problems? It turns out that the scale of magnitude of meta-gradient $\delta_t x_{i,t} h_{i,t}$ is different for different target output variances. In the bottom panel of Figure 5.2, we have plotted the trace of $|\delta_t x_{i,t} h_{i,t}|$ for $i = 0$ with the same setup as for the top panel. i.e., using the same meta-step-size values for the same problems. Values for problem 3 are shifted in time so that the values for problem 2 are visible. The values of $|\delta_t x_{i,t} h_{i,t}|$ for problem 2 and

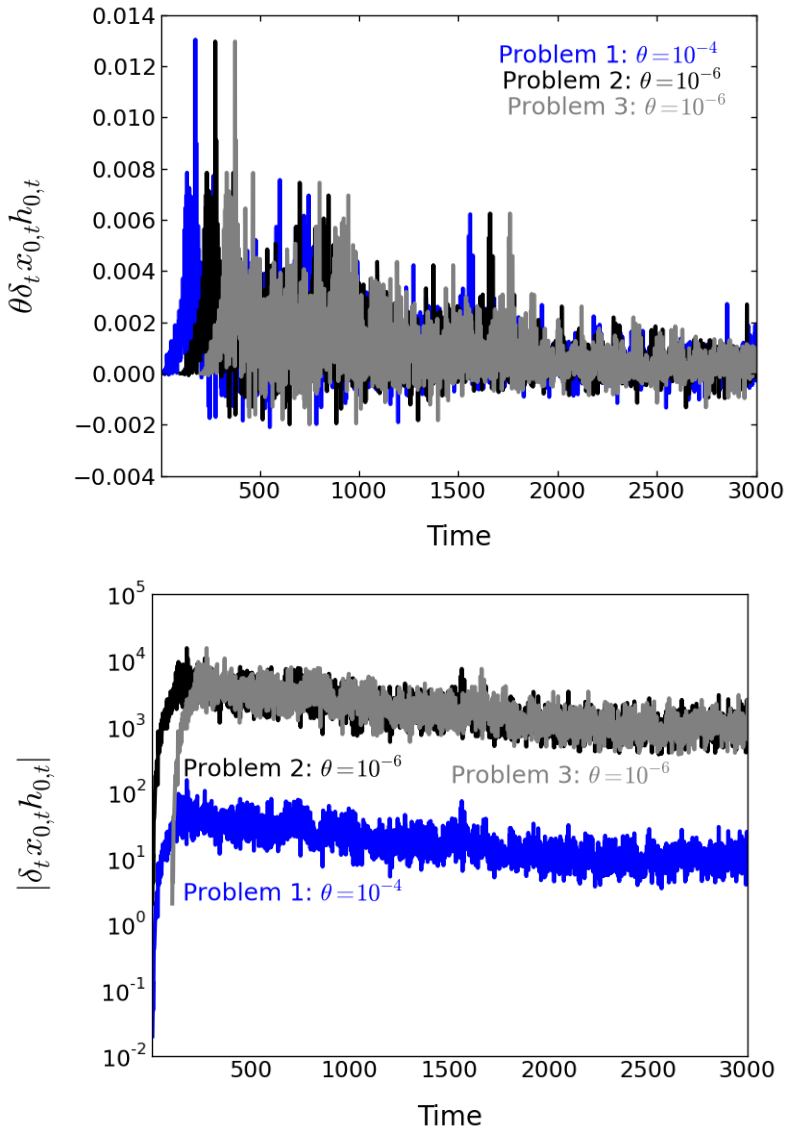

Figure 5.2: The traces of step-size update $\theta \delta_t x_{0,t} h_{0,t}$ (in top) and meta-gradient magnitudes $|\delta_t x_{0,t} h_{0,t}|$ (in bottom) for IDBD on the toy-test problems. For similar meta-step-size values, the step-size update values are the same for all problems, however the absolute values of meta-gradient are different in orders of magnitude, as shown in log scale.

3 are 100 times larger than that for problem 1. Therefore, the values of $\delta_t x_{i,t} h_{i,t}$ are affected by the target output variance, i.e., when the scale of the magnitude of the target weights or inputs are different in these problems. This is the reason the performance curves for different meta-step-size values and hence the best range of meta-step-size values are shifted 100 times in Figure 5.1.

We can view the step-size update term as $\theta u_{i,t}$ having two components: the update $u_{i,t}$ and the meta-step-size parameter θ, where $u_{i,t} = \delta_t x_{i,t} h_{i,t}$ in this case. When the scale of the magnitude of $u_{i,t}$ is increased, a lower value for the meta-step-size parameter needs to be chosen so that the scale of the magnitude of the update term $\theta u_{i,t}$ remains the same.

For a problem-independent choice of meta-step-size parameter, the order of the magnitude of θ should not change across problems. In order to be able to use the same values for θ, it is then necessary that the order of the magnitude for the update term $u_{i,t}$ remains unchanged across problems.

One way of achieving it is to normalize the update term by a measure which has the same orders of magnitude as $\delta_t x_{i,t} h_{i,t}$. However, the order of the magnitude of $\delta_t x_{i,t} h_{i,t}$ cannot be known a priori. A straight-forward way of achieving this normalization is to divide the update term by a running average of some norm of $\delta_t x_{i,t} h_{i,t}$ in the following way:

$$\begin{aligned} v_{i,t+1} &= v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}|^p - v_{i,t}), \\ \beta_{i,t+1} &= \beta_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}^{\frac{1}{p}}}, \end{aligned} \tag{5.2}$$

where p is some positive integer and $v_{i,t}$ is a running estimate of p norm of $\delta_t x_{i,t} h_{i,t}$. As the normalizer should be a positive quantity so that the approximate gradient direction is not altered by the normalization, the absolute value of $\delta_t x_{i,t} h_{i,t}$ is considered. This running estimate requires another parameter λ for discounting previous values of $\delta_t x_{i,t} h_{i,t}$. We multiply this parameter by the term $\alpha_{i,t} x_{i,t}^2$. As $\alpha_{i,t}$ itself is being adapted based on data, this running estimate is self regulated by the step-size element $\alpha_{i,t}$ and λ should not be different for different problems. By multiplying with $x_{i,t}^2$, the term $\alpha_{i,t} x_{i,t}^2$ becomes free of units, because the unit of $\alpha_{i,t}$ is the in-

verse of the squared inputs. It also prevents the running estimate to become small when the inputs are frequently zeros.

The simplest choice of p is 1, for which $v_{i,t}$ becomes a running average of the absolute values of $\delta_t x_{i,t} h_{i,t}$. The scale of the magnitude of this $v_{i,t}$ should be similar to $\delta_t x_{i,t} h_{i,t}$ and as the values vary for $\delta_t x_{i,t} h_{i,t}$ over time, the running estimate would also be able to track the scale. Therefore, by normalizing with $v_{i,t}$, the scale of the magnitude of the resulting normalized update term $\frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}$ should remain independent of the scale of the magnitude of any statistic of the problems.

If we choose $p = 2$, then $v_{i,t}$ becomes an estimate of the variance of $\delta_t x_{i,t} h_{i,t}$ and in the update rule of $\beta_{i,t}$, the update term is normalized by ${v_{i,t}}^{1/2}$, which is an estimate of the standard deviation. Such any p can be chosen and experimented to see the effect. However, all of them should ideally be able to achieve the problem independence in the scale of the magnitude of the update term.

In this work, we only consider $p = 1$ as the effect of the different choices of this parameter is empirically found to be insignificant.

This meta-normalization strategy can be applied both to IDBD and K1. In the following we describe the meta-normalization algorithm applied to IDBD, which we call *Meta-Normalized IDBD*:

Meta-Normalized IDBD :

$$
\begin{aligned}
v_{i,t+1} &= v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t}), \\
\beta_{i,t+1} &= \beta_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}, \\
\alpha_{i,t+1} &= e^{\beta_{i,t+1}}, \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\
h_{i,t+1} &= h_{i,t} \left(1 - \alpha_{i,t+1} x_{i,t}^2\right)^+ + \alpha_{i,t+1} \delta_t x_{i,t}.
\end{aligned}
$$

Similarly, this meta-normalization idea can be applied to K1:

Meta-Normalized K1:

$$
\begin{aligned}
v_{i,t+1} &= v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t}), \\
\beta_{i,t+1} &= \beta_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}, \\
b_{i,t+1} &= e^{\beta_{i,t+1}}, \\
\alpha_{i,t+1} &= \frac{b_{i,t+1}}{\hat{R} + \sum_j b_{j,t+1} x_{j,t}^2}, \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\
h_{i,t+1} &= (h_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}) \left(1 - \alpha_{i,t+1} x_{i,t}^2\right)^+ .
\end{aligned}
$$

We pursue detailed experimentation only on Meta-Normalized IDBD. We prevent numerical instability due to zero values in $v_{i,t+1}$ by not updating $\beta_{i,t}$ in those cases.

Our hypothesis regarding this algorithm is that, the choice of the meta-step-size parameter is less sensitive across problems than the original algorithm—IDBD—while retaining the effectiveness of it. It is the first algorithm along our journey toward achieving a completely automated step-size adaptation algorithm. It is interesting to know how far this algorithm can achieve toward this goal.

5.2 Results

In this section, we demonstrate the performance of Meta-Normalized IDBD on both toy test problems and robot problems. We focus on the robustness and sensitivity of this algorithm and does not explore the effect of the discounting factor λ here. We choose $\lambda = 10^{-2}$ for all the problems in this chapter and leave detailed experimentation with this parameter for later.

Figure 5.3 shows the results for the three toy problems. For Meta-Normalized IDBD, we used the same experimental setup as IDBD. The initial step-size value for problem 1, and 2 was $\frac{0.1}{20}$ and for problem 3, it was $\frac{0.1}{20 \times 100}$.

We find that Meta-Normalized IDBD is more robust across problems than IDBD in terms of the choice of the meta-step-size parameter. A value between 10^{-4} and

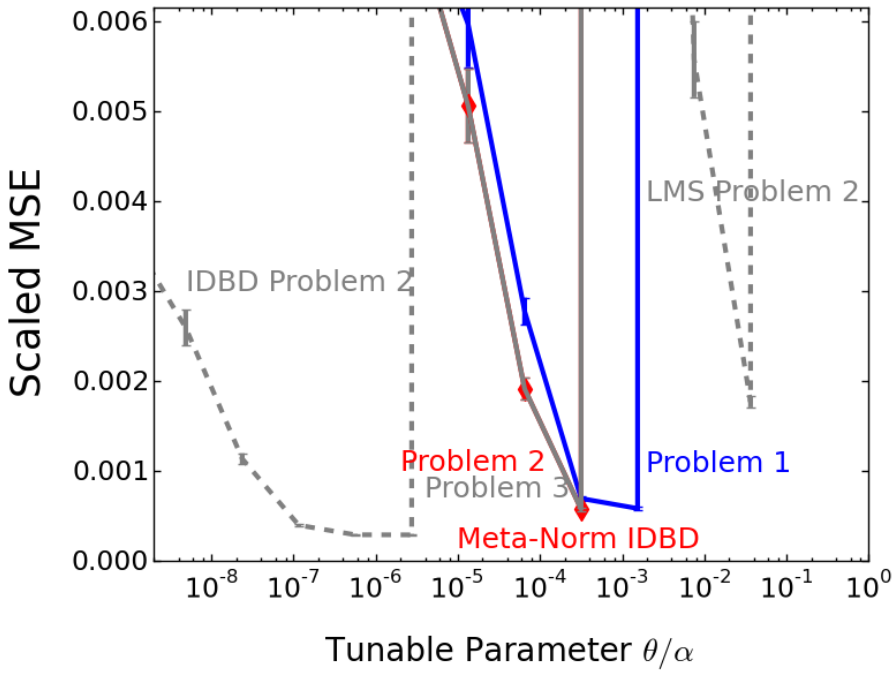

Figure 5.3: Performance vs. meta-step-size value for Meta-Normalized IDBD on toy problem 1, 2, and 3. Performance is scaled to superimpose the curves. Performance of LMS and IDBD on problem 2 is also given in dotted lines for comparison. A narrow range of meta-step-size values between 10^{-4} to 10^{-3} is best for Max-Normalized IDBD on all problems.

10^{-3} is found which can be used in all three problems and achieve almost as good performance as the best of IDBD.

However, the range of good values of the meta-step-size parameter is narrow for problem 2 and 3.

Next, we show the performance of Meta-Normalized IDBD on robot problems. Experimental setup of Meta-Normalized IDBD is the same as IDBD. Therefore, the initial step-size value was set to $\frac{0.0001}{57} = 1.754 \times 10^{-06}$.

Figure 5.4 shows the results of Meta-Normalized IDBD on the six robot problems. Performance of IDBD and LMS is also shown for comparison. In all these problems, the best performance of Meta-Normalized IDBD is at least as good as IDBD. Meta-Normalized IDBD is also more robust than IDBD as all of its best meta-step-size values are between 10^{-6} and 10^{-4}.

Figure 5.5 shows the performance of meta-norm IDBD for four robot problems and three of the toy problems in a scaled performance measure. MSE is scaled in

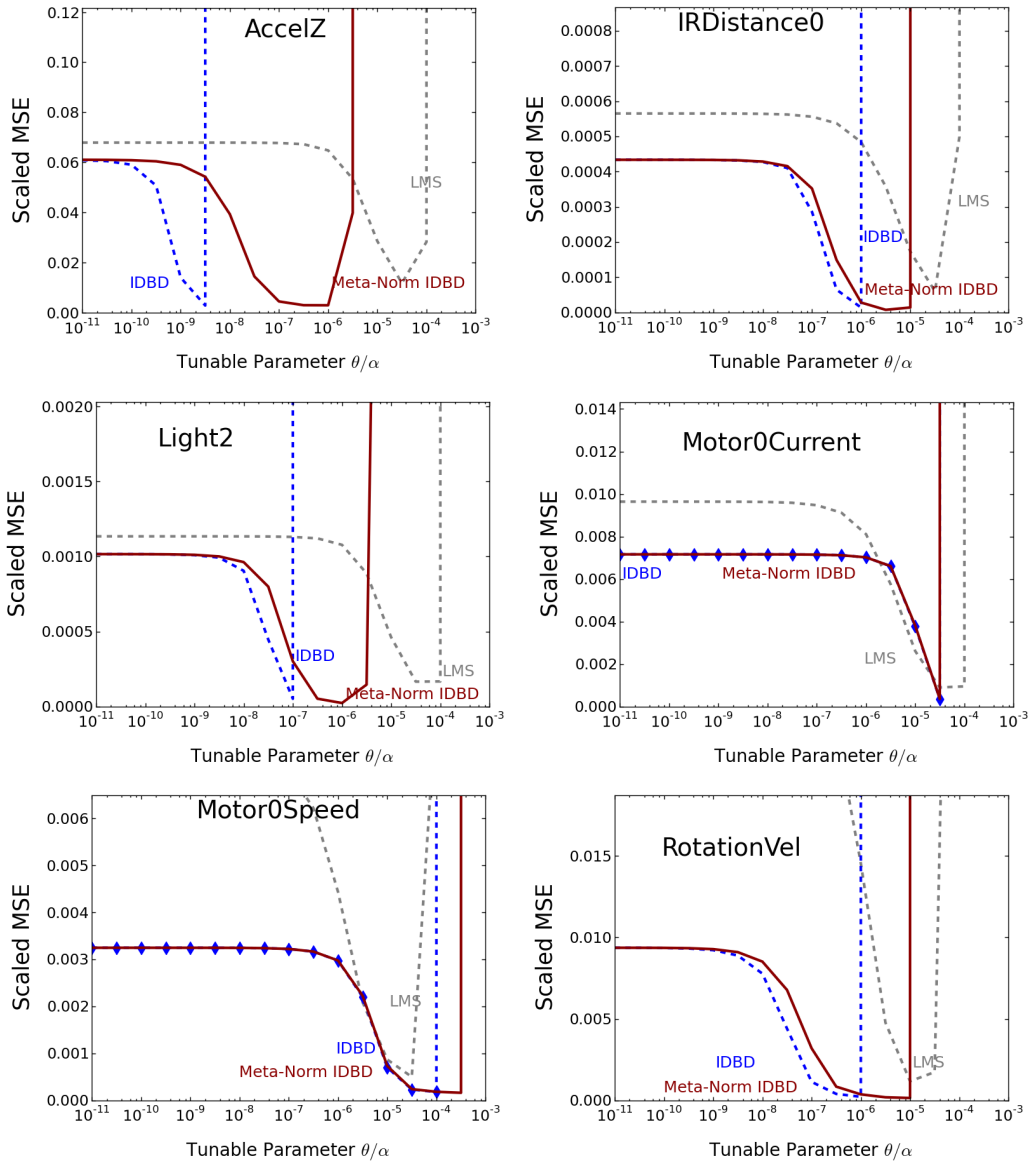

Figure 5.4: Performance vs. meta-step-size value for Meta-Normalized IDBD on six robot problems of predicting sensor signals in the next time step. Performance of IDBD and LMS is also given in dotted lines. Performance of Meta-Normalized IDBD is as good as IDBD.

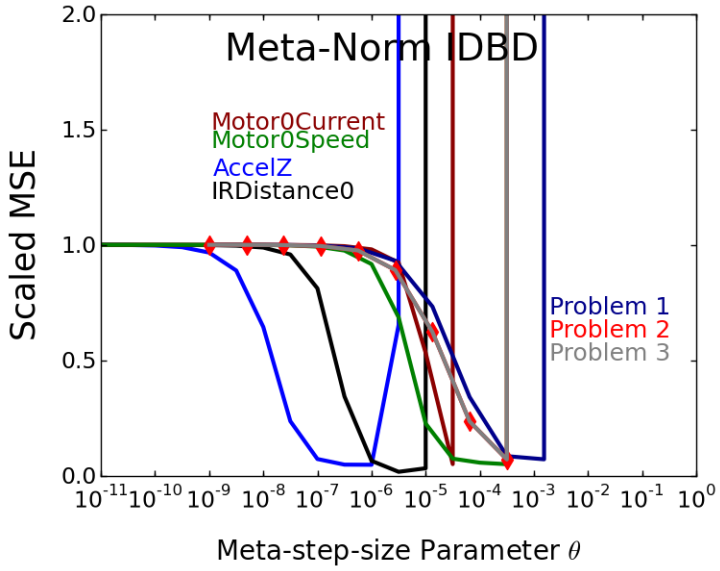

Figure 5.5: Meta-Normalized IDBD on three toy problems and six robot problems. There is no common range of meta-step-size values for Max-Normalized IDBD that is best on all problems. The best meta-step-size value also differs by orders of magnitude on robot problems.

a way that the performance for the smallest meta-step-size value for all the curves are one.

This figure shows that, although Meta-Normalized IDBD is more robust than IDBD, it is still not as robust as we would expect from an automatic step-size adaptation requiring no extensive manual tuning of the parameter. The best values of the meta-step-size parameter are different for different problems. The ranges of the best values are also narrow, requiring fine tuning.

We conclude these results by stating that meta-normalization strategy helped IDBD becoming more robust, however, it cannot still get rid of extensive parameter tuning.

5.3 Max Normalization

The fact that a single value or range of values for the meta-step-size parameter could not be achieved indicates that the scale of the magnitude of the update term $\delta_t x_{i,t} h_{i,t} / \left(v_{i,t+1}\right)$ is still affected by the variations in statistics, e.g., different target output variances across problems. Robot data is not as uniform or systematic as the

toy data and the variation in the best values of the meta-step-size parameter is more pronounced in the robot problems.

Extremely high values in signals, even if occurs rarely, can cause temporary high magnitude for the term $\delta_t x_{i,t} h_{i,t}$, which the running estimate $v_{i,t+1}$ may not take into consideration immediately to mitigate its effect on the stability of the step-size values. Such occurrence can be more common in real-world data. Therefore, normalization by a running estimate alone cannot prevent such instability. The meta-step-size value should be smaller at least temporarily to ensure stability in these cases. This affects the overall choice of the meta-step-size parameter. A usual value for this parameter that typically works in most situations leads to instability in such extreme cases.

A good way to prevent instability due to radical or extreme fluctuations in data is to upper-bound the magnitudes of the ratio $\delta_t x_{i,t} h_{i,t} / \left(v_{i,t+1}\right)$. Then the meta-step-size parameter does not need to be set to smaller values temporarily. Then one should be able to choose a suitable meta-step-size value which works for most typical cases and also does not result into instability in extreme cases. We upper bound the ratio to unity in the following way:

$$\begin{aligned} &\text{if } \frac{|\delta_t x_{i,t} h_{i,t}|}{v_{i,t+1}} > 1 : \\ &\qquad v_{i,t+1} = |\delta_t x_{i,t} h_{i,t}|, \end{aligned}$$

which is equivalent to writing

$$v_{i,t+1} = \max\left(v_{i,t+1}, |\delta_t x_{i,t} h_{i,t}|\right). \tag{5.3}$$

In fact, Equation (5.2) and (5.4) together can be written as

$$v_{i,t+1} = \max\left(|\delta_t x_{i,t} h_{i,t}|, v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t})\right)$$

We call the resulting algorithm—*Max-Normalized IDBD*. Each iteration of this algorithm is as follows:

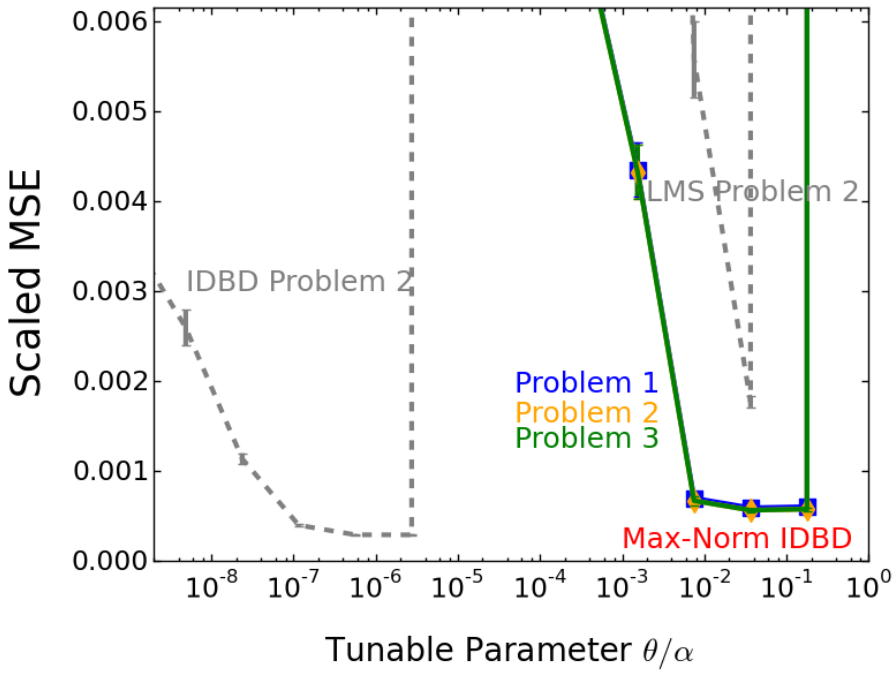

Figure 5.6: Performance vs. meta-step-size value for Max-Normalized IDBD on toy problem 1, 2, and 3. Performance is scaled to superimpose the curves. Performance of LMS and IDBD on problem 2 is also given in dotted lines for comparison. Max-Normalized IDBD with a meta-step-size value between 10^{-2} and 10^{-1} is best for all toy problems.

Max-Normalized IDBD :

$$v_{i,t+1} = \max\left(|\delta_t x_{i,t} h_{i,t}|, v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t})\right),$$
$$\beta_{i,t+1} = \beta_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}},$$
$$\alpha_{i,t+1} = e^{\beta_{i,t+1}},$$
$$w_{i,t+1} = w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t},$$
$$h_{i,t+1} = h_{i,t}\left(1 - \alpha_{i,t+1} x_{i,t}^2\right)^+ + \alpha_{i,t+1} \delta_t x_{i,t}.$$

A similar algorithm can also be devised for K1.

5.4 Results

In this section, we investigate the performance, robustness and sensitivity of Max-Normalized IDBD by testing it on the toy and robot problems. All the experimental

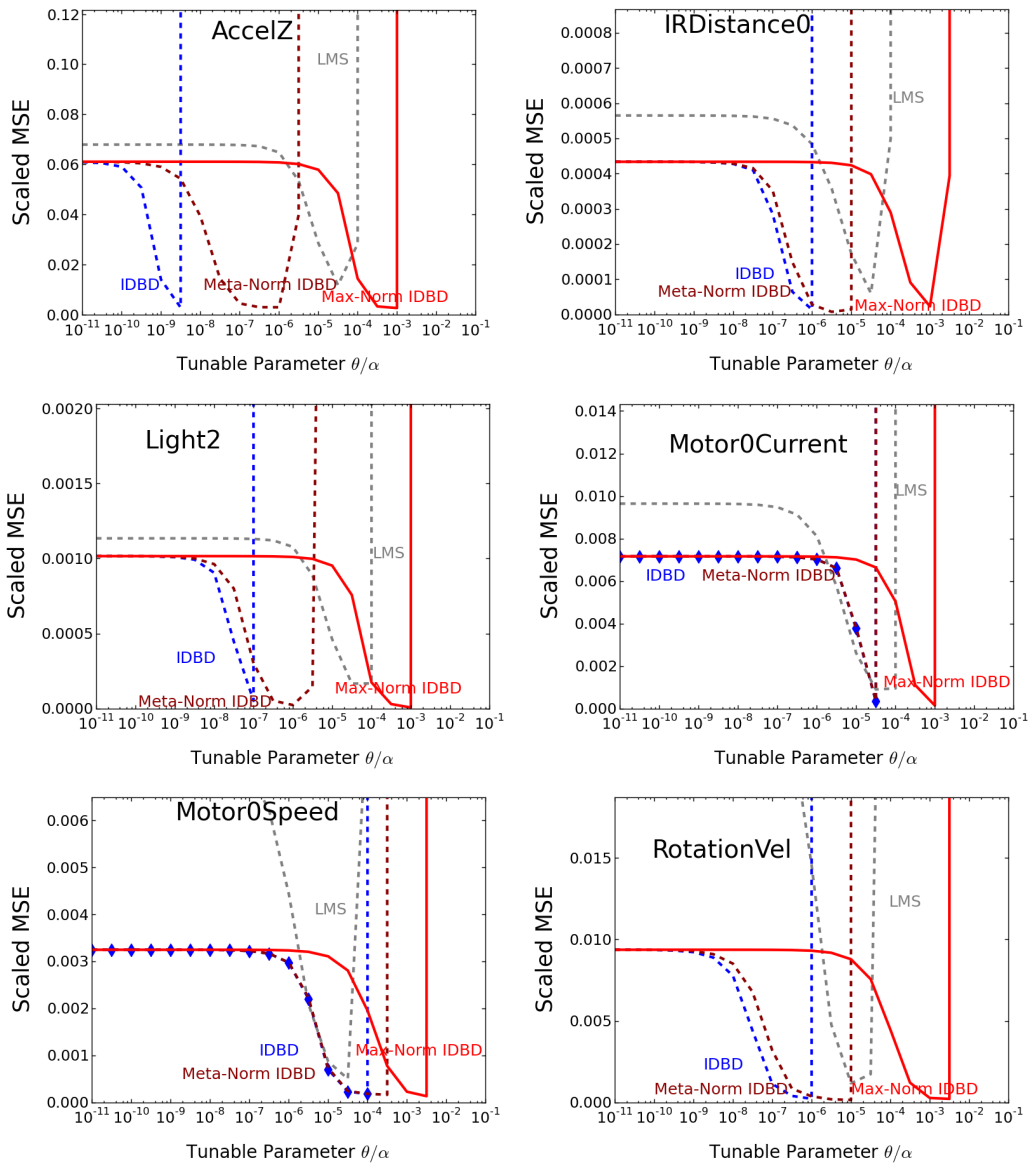

Figure 5.7: Max-Normalized IDBD on six robot problems of predicting sensor signals in the next time step. Performance of LMS, IDBD and Meta-Normalized IDBD is given in dotted lines for comparison. Performance of Meta-Normalized IDBD is as good as IDBD.

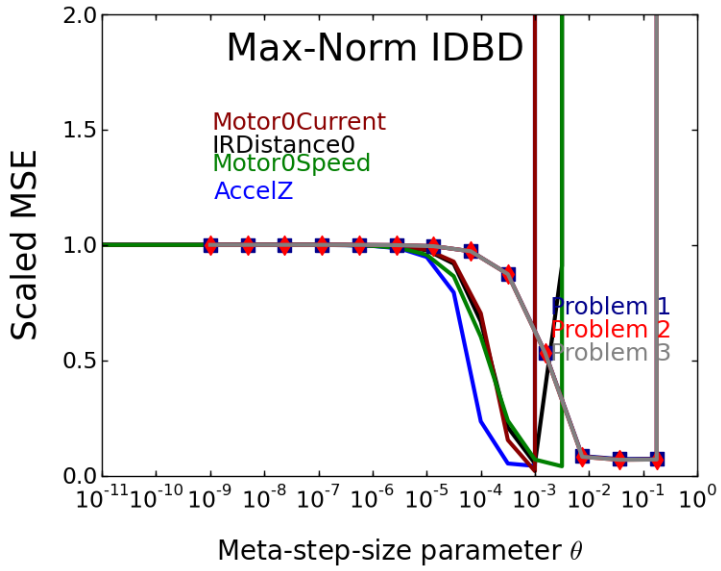

Figure 5.8: Max-Normalized IDBD on three toy problems and four robot problems. There is no common range of meta-step-size values for Max-Normalized IDBD that is best on all problems. There is, however, a common range of best meta-step-size values around 10^{-3} on all the robot problems.

setups are the same as IDBD on the same problems. The discounting factor λ was chosen to be 10^{-2}.

Max-Normalized IDBD is robust on the toy problems as shown in Figure 5.6. Three of the performance curves in scaled MSE overlap on each other completely. The range of the best values are between 10^{-2} and 10^{-1}. Max-Normalized IDBD with the meta-step-size parameter within this range of values is almost as good as the best performance of IDBD.

Figure 5.7 shows the results of Max-Normalized IDBD on the robot problems. We have also given the results of the Meta-Normalized IDBD, LMS and IDBD for comparison.

Max-Normalized IDBD is more robust than Meta-Normalized IDBD and performs best in all these problems.

Figure 5.8 shows the performance of Max-Normalized IDBD for four of the robot problems and three toy problems together.

This figure shows that Max-Normalized IDBD is robust up to some degree. But it indicates that Max-Normalized IDBD still needs some tuning of the meta-step-

size parameter. The range of the best values for the toy problems starts from about 10^{-2}, but Max-Normalized IDBD becomes unstable for this value in all the robot problems. Max-Normalized IDBD achieves its best performance for a value of the meta-step-size parameter around 10^{-3} on robot problems. In toy problems, MSE for the meta-step-size value of 10^{-3} is more than five times than the lowest MSE. The ranges of the best value of meta-step-size parameter for robot problems are also narrow.

5.5 Discussion

We have introduced new algorithms aiming at effective step-size adaptation that is more robust and less sensitive to the choice of the meta-step-size parameter than IDBD.

Our first algorithm is meta-noramalized IDBD. Here we normalize the step-size update term by a running estimate so that for the same meta-step-size value, the magnitude scale of the resulting update term does not change across problems. We provide insights on why the magnitude scale of the update term need not change in order to achieve a problem-independent meta-step-size parameter. This algorithm is more robust than IDBD while retaining superior performance of it. However, it is not less sensitive to the choice of the meta-step-size parameter within the same problem than IDBD. This algorithm is also not automatic enough to get rid of extensive manual tuning of its parameters. The best values of the meta-step-size parameter still varies by orders of magnitude between robot and toy problems.

We hypothesized that a single choice of the meta-step-size value for Meta-Normalized IDBD did not work in all the problems as for a sudden and infrequent increase in a statistic of the problem, the meta-step-size value may need to be smaller to ensure stability. This makes the choice of the meta-step-size parameter problem dependent.

Based on Meta-Normalized IDBD, we then introduced Max-Normalized IDBD where the update term cannot be affected arbitrarily by a sudden increase in the statistics of the problem. The update term is upper bounded in this algorithm.

Max-Normalized IDBD is more robust than Meta-Normalized IDBD and performs almost as well as IDBD. However, performance of Max-Normalized IDBD is as sensitive as IDBD and Meta-Normalized IDBD on a same problem. It also requires some tuning of the meta-step-size parameter. A common range of meta-step-size values was not found for which this algorithm achieves its best performance on all problems.

If Max-Normalized IDBD was even less sensitive to the choice of the meta-step-size parameter, e.g., if the range of the best values for the robot problems was extended toward the higher meta-step-size values, a common range of best values could have been found.

Another problem-dependence that still remains in these algorithms is with the initial step-size value. Three different scales of values were used in our experiments. In problem 1 and 2, the initial step-size value is 5×10^{-3}, for problem 3, it is 5×10^{-5} and for the robot problems it is about 1.75×10^{-6}. Its sensitivity with respect to the initial choice of the step-size value is in fact as much as IDBD.

Max-Normalized IDBD is the algorithm of choice among the two we have introduced here. To achieve an automatic step-size adaptation, it still needs to be more robust and less sensitive with respect to the choice of its meta-step-size parameter.

Chapter 6

Pre-normalization in Step Size

In this chapter we introduce a new modification to our Max-Normalized IDBD algorithm, developed in Chapter 5. Our goal of step-size adaptation in terms of robustness is to find a common range of meta-step-size values that can achieve best performance in all of the toy and robot problems. Here, we aim at developing an algorithm that is more robust and less sensitive than Max-Normalized IDBD.

To achieve a more robust algorithm, we deploy a strategy of normalizing the step size that helps in keeping its values stable for higher meta-step-size values. It is applied without using the knowledge of the sample error or target output value and hence we call this *pre-normalization*. We will find through experiments on toy and robot problems that this new modification helps the step-size adaptation algorithm to become more robust.

6.1 The Technique of Pre-normalization

Why does the step-size adaptation in Max-Normalized IDBD become unstable for higher values of the meta-step-size parameter? The range of the best values of the meta-step-size parameter for Max-Normalized IDBD is narrow. When a value slightly larger than the values in that range is chosen, performance worsens considerably. Large meta-step-size values result in larger changes in the step-size parameter. The algorithm can become sensitive due to such large changes in the step-size parameter and may also result into instability of the algorithm. If we do not let the step-size parameter grow larger, a large meta-step-size value would not be able to

let the algorithm become unstable.

In deterministic problems where the gradient in known exactly, a sufficient condition on the step-size parameter for the stability of the algorithm can be easily found. For example, if the step-size values at each time step are chosen in a way that the successive gradients have a positive inner product, then the algorithm cannot diverge. In stochastic problems, where only a noisy gradient estimate is available at each time step, such conditions on the successive exact gradients cannot be ensured. Albeit, a similar idea can be used to keep the step-size parameter from growing large.

Stochastic gradient descent such as LMS can be thought as a gradient descent in a surface based on the single sample considered at each time step. This surface is defined as a quadratic function $\left(y_t - \boldsymbol{w}^\top \boldsymbol{x}_t\right)^2$ of $\boldsymbol{w}$. This is also the sample squared error. A Gradient on this surface at the current weight vector $\boldsymbol{w}_t$ is $\nabla_{\boldsymbol{w}} \left(y_t - \boldsymbol{w}^\top \boldsymbol{x}_t\right)^2 \big|_{\boldsymbol{w}=\boldsymbol{w}_t}$ or can be simply written as $\nabla_{\boldsymbol{w}_t} \left(y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right)^2 = \nabla_{\boldsymbol{w}_t} \delta_t^2$. We can define largeness of the step-size value based on this surface so that it can prevent the step-size value from being unstable in the global performance surface.

Adaptation using a vector step-size parameter scales the sample gradient direction $-\delta_t \boldsymbol{x}_t$ to a direction $-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$, which is no longer a gradient direction if the elements of the step-size vector have different values. However, it is a descent direction in the surface of the sample squared error, because this direction has a positive inner product with the gradient direction, as shown below:

$$
\begin{aligned}
&\left(-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)\right)^\top (-\delta_t \boldsymbol{x}_t) \\
&= \delta_t^2 \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top (\boldsymbol{x}_t) \\
&= \delta_t^2 \sum_j \alpha_{j,t+1} x_{j,t}^2 \\
&\geq 0,
\end{aligned}
$$

given that the elements of step-size $\boldsymbol{\alpha}_t$ are non-negative. This new direction vector $-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$ can be an improvement over the gradient direction in a global sense.

If the weight moves too much along the direction vector $-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$, it may start to diverge. A linear scaling of this direction vector is necessary when the step

toward this direction is large.

We can scale down the direction vector $-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$ linearly to ensure that the gradient in the current weight value $\boldsymbol{w}_t$ and the gradient in the updated weight value $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$ have a nonnegative inner product:

$$\left(\nabla_{\boldsymbol{w}_{t+1}} \left(y_t - \boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t\right)^2\right)^\top \left(\nabla_{\boldsymbol{w}_t} \left(y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right)^2\right) \geq 0. \tag{6.1}$$

If the elements of the step-size vector is arbitrarily large, this condition cannot be true.

The expression in (6.1) can be simplified in the following way:

$$\begin{aligned}
\left(\nabla_{\boldsymbol{w}_{t+1}} \left(y_t - \boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t\right)^2\right)^\top \left(\nabla_{\boldsymbol{w}_t} \left(y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right)^2\right) &\geq 0, \\
\left(-2\left(y_t - \boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t\right)^\top \left(-2\left(y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t\right) &\geq 0, \\
4\left(y_t - \left(\boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)\right)^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t^\top (\delta_t \boldsymbol{x}_t) &\geq 0, \\
\delta_t \left(y_t - \boldsymbol{w}_t^\top \boldsymbol{x}_t - \delta_t \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t^\top \boldsymbol{x}_t &\geq 0, \\
\delta_t \left(\delta_t - \delta_t \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t^\top \boldsymbol{x}_t &\geq 0, \\
\delta_t^2 \left(1 - \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t\right)\boldsymbol{x}_t^\top \boldsymbol{x}_t &\geq 0, \\
\left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t &\leq 1, \\
\text{or equivalently } \sum_j \alpha_{j,t+1} x_{j,t}^2 &\leq 1.
\end{aligned} \tag{6.2}$$

When this condition is false, the step-size value causes the update of the weight vector to overshoot the minimum point of the surface along the direction $-\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$.

We detect a step-size parameter to be large whenever the condition in Equation (6.2) is false. If a large step-size parameter is detected, i.e., condition $\sum_j \alpha_{j,t+1} x_{j,t}^2 \leq 1$ is false, all the step-size elements should then be reduced in a way that the condition in Equation (6.2) is true again. All the elements of the vector step-size parameter should be reduced by the same scalar amount to retain the direction given by the update term $-\boldsymbol{\alpha}_{t+1} \circ (\delta \boldsymbol{x}_t)$.

Let us normalize the step-size vector by a scalar quantity $c(t)$ at each time step t to guarantee the condition in Equation (6.2). If we repeat the above derivation by using the definition of the updated weight as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1}{c(t)}\boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$ instead of $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{\alpha}_{t+1} \circ (\delta_t \boldsymbol{x}_t)$, then it follows immediately that the condition on $c(t)$ is

$$\begin{aligned} c(t) &\geq \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t, \\ \text{or equivalently } c(t) &\geq \sum_j \alpha_{j,t+1} x_{j,t}^2. \end{aligned}$$

We choose, $c(t) = \sum_j \alpha_{j,t+1} x_{j,t}^2$ and consider the new step-size value to be $\alpha_{i,t+1} = \alpha_{i,t+1}/c(t)$ whenever the condition $\sum_j \alpha_{j,t+1} x_{j,t}^2 \leq 1$ is false. It can be easily verified that by doing so, the condition $\sum_j \alpha_{j,t+1} x_{j,t}^2 \leq 1$ becomes true again.

This conditional normalization of the step-size vector is used before the weight is updated in Max-Normalized IDBD:

$$\begin{aligned} &\text{if } \sum_j \alpha_{j,t+1} x_{j,t}^2 > 1 : \\ &\qquad \alpha_{i,t+1} = \frac{\alpha_{i,t+1}}{\sum_j \alpha_{j,t} x_{j,t}^2}, \\ &w_{i,t+1} = w_{i,t} + \alpha_{i,t+1} \delta_t x_{i,t}, \\ &h_{i,t+1} = h_{i,t} \left(1 - \alpha_{i,t+1} x_{i,t}^2\right) + \alpha_{i,t+1} \delta_t x_{i,t}. \end{aligned} \tag{6.3}$$

As the step-size parameter is changed, the auxiliary variable $\beta_{i,t}$ in the next time step will no longer correspond to the logarithm of $\alpha_{i,t}$. Hence, following modifications on the $\beta_{i,t}$ update rule is used in Max-Normalized IDBD:

$$\begin{aligned} \beta_{i,t+1} &= \log \alpha_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}, \\ \alpha_{i,t+1} &= e^{\beta_{i,t+1}}, \end{aligned}$$

i.e., $\log \alpha_{i,t}$ is used instead of $\beta_{i,t}$.

The truncation $[v]^+$ for $\left(1 - \alpha_{i,t+1} x_{i,t}^2\right)$ in the update rule of $h_{i,t}$ in Equation (6.3) is not used as this term here is never negative now. This follows directly from satisfying the condition in Equation (6.2).

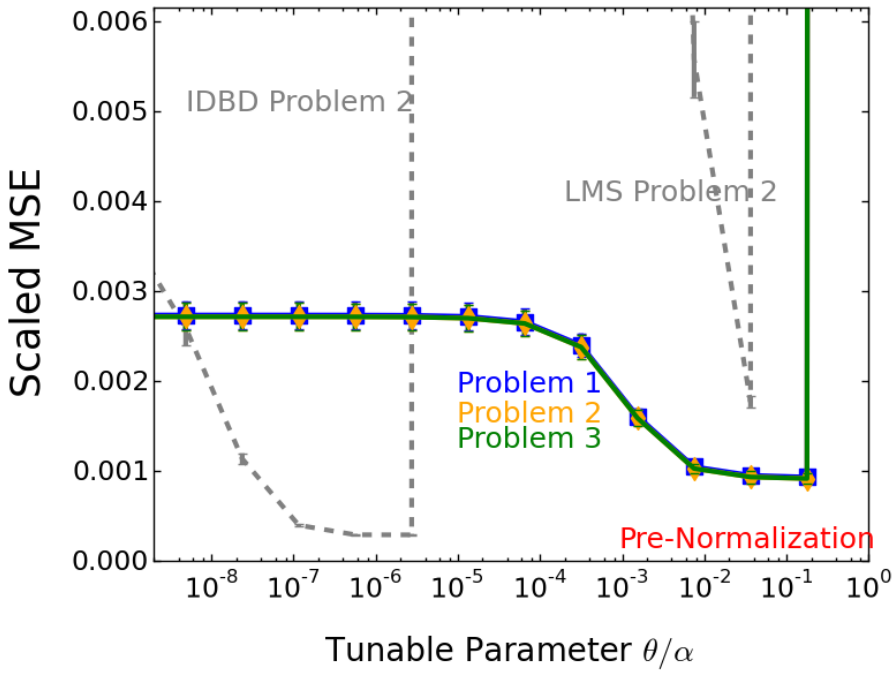

Figure 6.1: Performance vs. meta-step-size value for Pre-normalization technique applied to Max-Normalized IDBD on toy problem 1, 2, and 3. Performance is scaled to superimpose the curves. Performance of LMS and IDBD on problem 2 is also given in dotted lines for comparison. Meta-step-size value between 10^{-2} and 10^{-1} is best on all problems.

As this normalization is done before weight update and without using the knowledge of the sample error or the target output, we call it *pre-normalization* technique. Pre-normalization technique ensures that the updated weight does not overshoot the minimum of its immediate loss function δ_t^2 in a single time-step.

This pre-normalization technique is useful to detect a large step-size at every time-step. Therefore, this strategy may help the algorithm remain stable whenever the step-size parameter increases abruptly due to a large meta-step-size value.

6.2 Results

We experiment with this pre-normalization technique applied to Max-Normalized IDBD and find its robustness and sensitivity in toy and robot problems. We choose $\lambda = 10^{-2}$ and the initial step-size value to be 0.1 in all the experiments in this chapter.

This pre-normalization technique applied to Max-Normalized IDBD is more

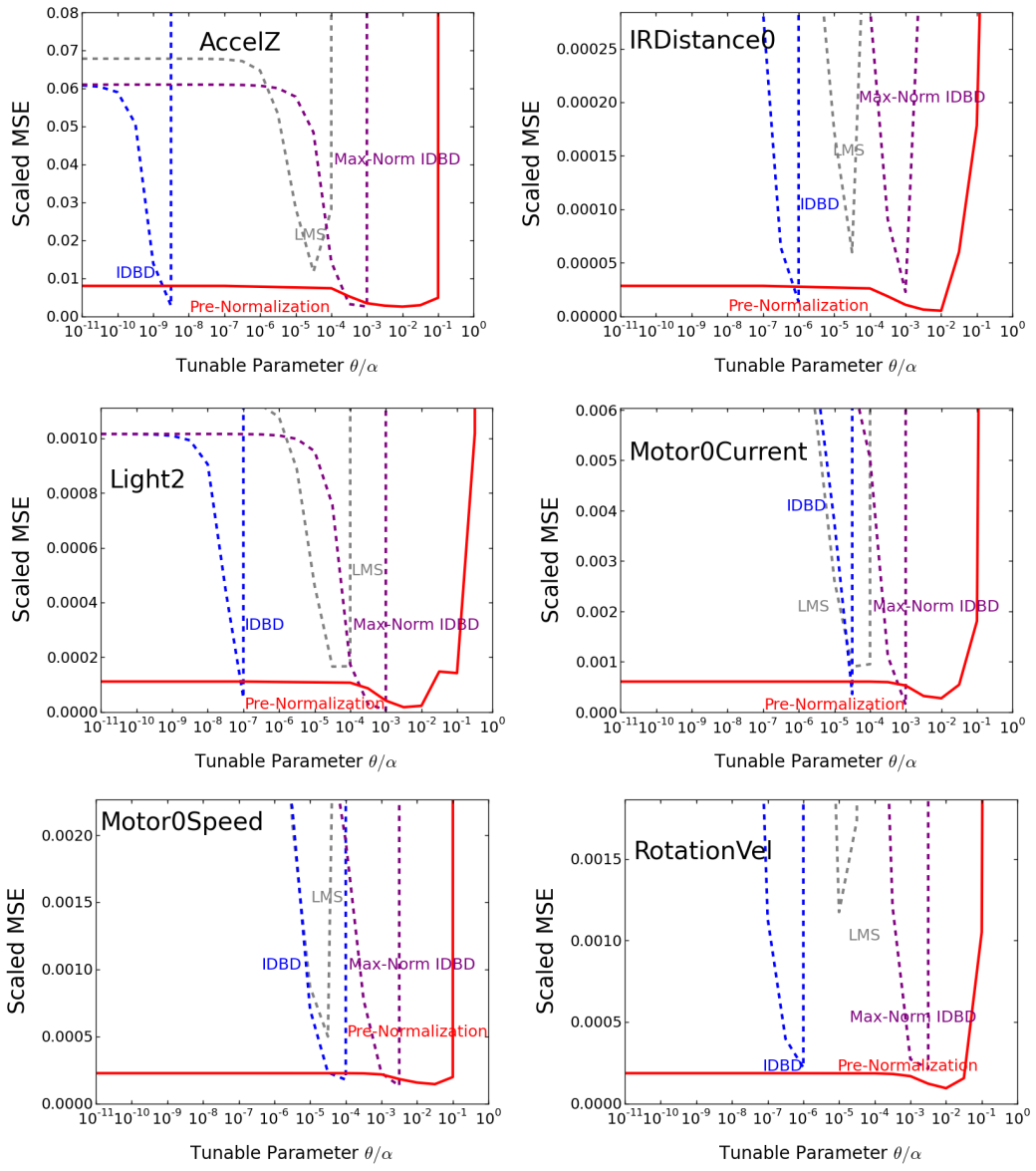

Figure 6.2: Performance vs. meta-step-size value for Pre-normalization applied to Max-Normalized IDBD on six robot problems of predicting sensor signals in the next time step. Performance of LMS, IDBD and Max-Normalized IDBD is given in dotted lines for comparison. Pre-normalized algorithm is less sensitive than IDBD and Max-Normalized IDBD. Performance of it is as good as IDBD in most of the problems and better in problem IRDistance0 and RotationVel.

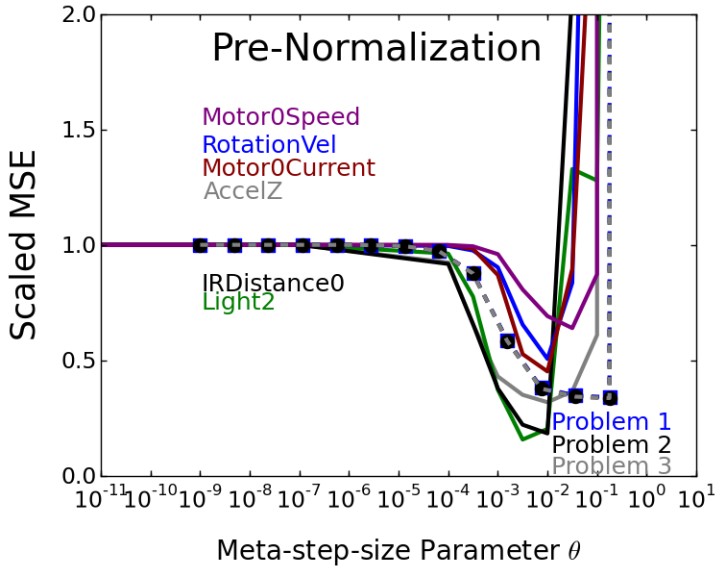

Figure 6.3: Performance vs. meta-step-size value for pre-normalization applied to Max-Normalized IDBD on three toy problems and six robot problems. For the meta-step-size value around 10^{-2}, this algorithm achieves its best performance on most of the problems.

robust and less sensitive than IDBD on the toy problems as shown in Figure 6.1. The range of the best meta-step-size values is between 10^{-2} and 10^{-1}. Performance for the smallest meta-step-size value shown at the leftmost side of the performance curve is also not more than three times of the best performance. It means performance is less sensitive to the choice of meta-step-size value comparing to IDBD. However, this is achieved at the expense of increasing MSE. Best performance of pre-normalization technique is worse than that of IDBD, although it is better than LMS.

This pre-normalization technique makes Max-Normalized IDBD less sensitive to the meta-step-size parameter for the robot problems, as shown in Figure 6.2. All of the curves are flatter than IDBD and Max-Normalized IDBD. For many of these problems, performance for the smallest meta-step-size value is almost as good as the best performance of IDBD.

In robot problems, the best performance of this pre-normalization technique is at least as good as that of IDBD. In problems IRDistance0 and RotationVel, its performance is better than IDBD.

Finally, we summarize the results on all the toy and robot problems on a scaled

performance measure in Figure 6.3. MSE is scaled in a way that the performance for the smallest meta-step-size value for all the curves are one.

This pre-normalization technique is the most robust among all the algorithms that we have experimented so far. Near best performance is achieved for a meta-step-size value around 10^{-2} on all these problems. A meta-step-size value between 10^{-3} and 10^{-2} is safe and adapts the step-size parameter effectively in all these problems. On some of the problems a meta-step-size value less than 10^{-2} may not be the best, however it can still achieve superior performance than LMS in all the problems.

6.3 Discussion

The pre-normalization technique applied to Max-Normalized IDBD has performed most robustly among all the algorithms that we have experimented with. We have devised this technique so that abrupt increases in the step-size parameter due to large meta-step-size values can be detected and reduced to ensure stability. This technique could indeed extend the range of the best meta-step-size values for Max-Normalized IDBD toward higher values.

There is a common range of meta-step-size values around 10^{-2} that achieves near-best performance in all the problems that we have demonstrated. This pre-normalization based algorithm is the only one for which a common range of good meta-step-size values is found.

This algorithm is also less sensitive than our previous algorithms and the existing ones. Performance for the smallest meta-step-size value such as 10^{-11} is better than the best of LMS in all cases.

This pre-normalization technique has also achieved robustness with respect to the initial step-size value. We have used an initial step-size value of 0.1 on all of the toy and robot test problems.

Chapter 7

Autostep

This chapter summarizes our final algorithm—Autostep—that puts all of our algorithmic contributions together. Autostep generalizes the step-size adaptation algorithm based on pre-normalization that we have introduced in Chapter 6. Autostep combines three modifications on IDBD: normalization in the step-size update, upper bound on the step-size update and reducing the step-size value when detected to be large.

Autostep is the result of applying our last algorithmic technique presented in Chapter 6. All the algorithmic ideas are again briefly described here and the description of the full algorithm is given in this chapter. Autostep has three parameters, the discounting factor, the meta-step-size parameter and the initial step-size value. Here, we empirically demonstrate on the toy and robot test problems that performance of Autostep is robust on the choice of all these parameters.

Our empirical results demonstrate that by developing Autostep we satisfactorily achieve most of our goals of automatic step-size adaptation in linear supervised setting. Since this final algorithm already appears in the previous chapter, some of the results of empirical experiments are also presented there. In this chapter we rigorously test Autostep for different parameters it contains. Most importantly, we test Autostep on six new robot problems in this chapter. We already gain confidence on our final algorithm in the previous chapter by empirically testing it over the three toy problems and the six robot problems. In that sense, these nine problems can be considered as the training set for the Autostep because these problems were introduced before developing Autostep. On the other hand, we gain more confidence on

the robustness of Autostep by testing it on the six new robot problems. These six new robot problems are introduced after Autostep is developed and hence can be considered as the test set for Autostep.

7.1 The Autostep Algorithm

IDBD can adapt the step-size parameter, however the effectiveness of adaptation or even stability depends on its meta-step-size parameter. Following is the IDBD algorithm:

$$\begin{aligned}
\beta_{i,t+1} &= \beta_{i,t} + \theta\delta_t x_{i,t} h_{i,t}, \\
\alpha_{i,t+1} &= e^{\beta_{i,t+1}}, \\
w_{i,t+1} &= w_{i,t} + \alpha_{i,t+1}\delta_t x_{i,t}, \\
h_{i,t+1} &= h_{i,t}\left(1 - \alpha_{i,t+1}x_{i,t}^2\right)^+ + \alpha_{i,t+1}\delta_t x_{i,t},
\end{aligned} \tag{7.1}$$

where $x_{i,t}$ is the ith element of input vector, y_t is a scalar, target output, $w_{i,t}$ is the ith element of the estimated weight vector, $\alpha_{i,t+1}$ is the ith element of step-size vector being adapted, $\beta_{i,t}$ is the ith element of a vector that is the logarithm of the step-size element $\alpha_{i,t}$ and updated using a gradient-descent based rule and $\theta > 0$ is the meta-step-size parameter. Extra memory $h_{i,t}$ is an element of a vector incrementally updated in order to approximate the gradient update of $\beta_{i,t}$.

There is no straightforward way of setting the meta-step-size parameter. In Chapter 3 and 4 we have shown that an appropriate meta-step-size value is different in orders of magnitude in different problems. We have introduced a series of techniques in this work so that manual tuning of this parameter can be avoided, i.e., the same algorithm can be used on various problems without changing its setting of parameters.

Our first technique, introduced in Chapter 5, was to normalize the step-size update $\theta\delta_t x_{i,t} h_{i,t}$ in Equation (7.1) by a running average of the absolute value of $\delta_t x_{i,t} h_{i,t}$:

$$v_{i,t+1} = v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t}), \tag{7.2}$$

$$\beta_{i,t+1} = \beta_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}. \tag{7.3}$$

The meta-step-size parameter θ in IDBD has the same unit as the inverse of $\delta_t x_{i,t} h_{i,t}$ and this normalization makes the meta-step-size parameter θ in Equation (7.3) free of units. However, it does not make θ completely problem independent. The choice of this parameter can still be affected by problem-dependent statistics when occasional but extreme changes occur in the value of target output or inputs.

Our second technique is to modify the normalizer $v_{i,t+1}$ in order to upper bound the update term $\delta_t x_{i,t} h_{i,t} / v_{i,t+1}$, by replacing the update in Equation (7.2) with the following:

$$v_{i,t+1} = \max \left(|\delta_t x_{i,t} h_{i,t}|, v_{i,t} + \lambda \alpha_{i,t} x_{i,t}^2 (|\delta_t x_{i,t} h_{i,t}| - v_{i,t}) \right).$$

It is introduced in Chapter 5. By using this modification, the term $\delta_t x_{i,t} h_{i,t} / v_{i,t+1}$ is essentially upper bounded to one. This way, the choice of θ is guarded against the effect of occasional and abrupt increases in the value of $\delta_t x_{i,t} h_{i,t}$.

Our last technique is to detect a high step along the direction of weight update in the surface of the sample squared error and scale the step-size vector to smaller values so that updated weight vector does not overshoot the minimum in the surface along the update direction. We detect such large step sizes and reduce it as follows:

$$\text{if } \sum_j \alpha_{j,t+1} x_{j,t}^2 > 1 :$$

$$\alpha_{i,t+1} = \frac{\alpha_{i,t+1}}{\sum_j \alpha_{j,t} x_{j,t}^2}.$$

To transmit this updated step-size values into the next time step through $\beta_{i,t}$, the update rule of $\beta_{i,t}$ is modified in the following way:

$$\beta_{i,t+1} = \log \alpha_{i,t} + \theta \frac{\delta_t x_{i,t} h_{i,t}}{v_{i,t+1}}.$$

Initialization:

$\lambda \leftarrow 10^{-4}$ or a value less than 1

$\theta \leftarrow 0.01$

$h_i \leftarrow v_i \leftarrow 0$

$\alpha_i \leftarrow 0.1$ or a higher value

for each new data sample $(x_1, \ldots, x_n, y)$ **do**

$\delta \leftarrow y - \sum_{i=1}^{n} w_i x_i$

for $i = 1, \ldots, n$ **do**

$v_i \leftarrow \max\left(|\delta x_i h_i|, v_i + \lambda \alpha_i x_i^2 \left(|\delta x_i h_i| - v_i\right)\right)$

if $v_i \neq 0$ **then**

$\alpha_i \leftarrow \alpha_i \exp\left(\theta \frac{\delta x_i h_i}{v_i}\right)$

end if

end for

$m \leftarrow \sum_{i=1}^{n} \alpha_i x_i^2$

if $m > 1$ **then**

for $i = 1, \ldots, n$ **do**

$\alpha_i \leftarrow \frac{\alpha_i}{m}$

end for

end if

for $i = 1, \ldots, n$ **do**

$w_i \leftarrow w_i + \alpha_i \delta x_i$

$h_i \leftarrow h_i \left(1 - \alpha_i x_i^2\right) + \alpha_i \delta x_i$

end for

end for

Figure 7.1: The Autostep Algorithm

This technique is introduced in Chapter 6. Due to this modification, large and detrimental increases of the step-size parameter is guarded.

All these techniques can be used separately on IDBD. However, we found that using three of them together achieves us the most robust algorithm.

Our resulting algorithm—Autostep—is given in Figure 7.1.
This algorithm has three parameters: meta-step-size θ, discounting factor λ and initial step-size vector $\boldsymbol{\alpha}_0$. This algorithm requires $4n$ memory, where n is the size of the input vector.

In Chapter 6 we have shown that this algorithm for $\lambda = 10^{-2}$ is robust on the choice of the meta-step-size parameter. A common range of meta-step-size values around 10^{-2} exists, for which best performance can be achieved on all of our test problems. No other algorithm can perform similarly.

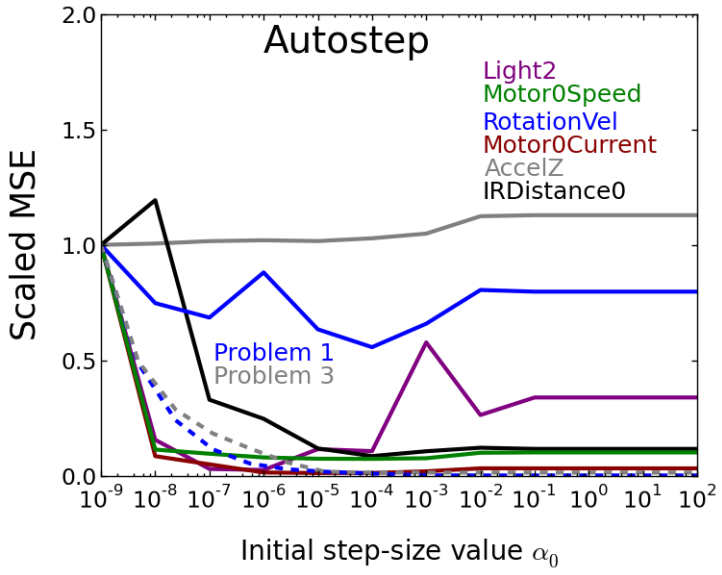

Figure 7.2: Performance of Autostep for different initial step-size values is shown on two toy problems and six robot problems. Curves for toy problems are in dotted lines. For an initial step-size value more than 10^{-2}, Autostep achieves minimum MSE in most of these problems.

Autostep is also robust on the choice of the initial step-size value. We have used initial step-size value to be 0.1 on all the toy and robot problems in the Chapter 6 and found it to be effectively adapting the step-size parameter.

The modifications of Autostep can also be applied to K1. It is not clear whether the conditional pre-normalization of the step-size values should be applied to K1, because a form of normalization is already used in K1. Our preliminary results applying first two modification techniques on K1 show that the resulting algorithm is also robust and less sensitive to the choice of the meta-step-size parameter, while retaining the superior performance of K1. However, we did not conduct thorough experimentation of this algorithm and hence do not provide the experimental results.

In the following, we thoroughly explore the effect of the choice of the initial step-size value on Autostep. We also experiment with Autostep to find how robust this algorithm is on the choice of the parameter λ.

7.2 Tests for Initial Step-Size Values

In this section, we empirically find how robust Autostep is on its initial step-size values. We have varied the initial step-size values of Autostep for all the test problems. The description and setup of the toy and robot test problems are given in Chapter 3 and Chapter 4, respectively. We have chosen meta-step-size $\theta = 10^{-2}$ as this value is found to achieve best performance in all of the problems in the results of Chapter 6. We have chosen $\lambda = 10^{-2}$ following the setup in Chapter 6. We show the results for all the problems in Figure 7.2. The MSE vs. initial step-size value curves are superimposed in a single figure by aligning the smallest initial step-size values (10^{-9} here) for each curve together. MSE is calculated by averaging over 30 different runs.

There is a common range—values greater than 10^{-2}—for which near-best performance is achieved on all problems. There are some fluctuations in values for Light2 and RotationVel. However, differences in the fluctuated values are not significant in that the standard error is high for those values. Initial step-size value can be chosen as large as 100 in all these problems and still achieve near-best performance.

No initial step-size values were found for which Autostep diverges in any of these problems.

The pre-normalization technique, where large step-size values are reduced, helps to find a good step-size value quickly. If large initial step-size value is chosen, it is more likely that it will be reduced by pre-normalization quickly. On the other hand, if a small initial step-size value is chosen, it may take some time before pre-normalization is enacted. Therefore, small initial step-size value can be slow to adapt to correct values. We recommend using an initial step-size value 0.1 or more.

7.3 Tests for λ

In this section we test whether the robustness and sensitivity of Autostep with respect to the meta-step-size parameter is affected by different choices of λ. We experiment wih five different combinations of these values: we have chosen $\lambda =$

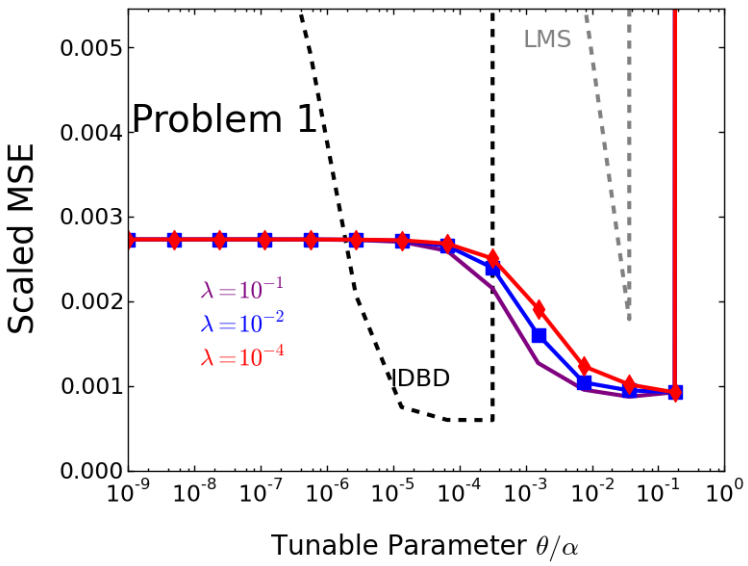

Figure 7.3: Performance vs. meta-step-size value for Autostep with three different settings of λ on toy problem 1. Performance is scaled to superimpose the curves. Performance of IDBD and LMS is also given in dotted lines. Curves do not shift in an order of magnitude for different choices λ.

$10^{-1}, 10^{-2}, 10^{-4}$. Initial step-size value is chosen to be 0.1. For each case, we have varied the meta-step-size θ.

Figure 7.3 shows performance of these five variants of Autostep on toy problem 1. These curves are very close to each other. Robustness and sensitivity of Autostep does not seem to be affected by the choice these parameter values.

Figure 7.4 shows performance of the three of Autostep with different λ values on robot problems. Curves are close to each other in all these problems. There is a slight shift of the curves toward higher meta-step-size values for smaller values of λ. For smaller values of λ, the normalizer estimates the average slowly and hence it is less sensitive to the changes in recent values. Therefore, Autostep with smaller λ values can remain stable for higher meta-step-size values.

7.4 Summary Results

In this section we summarize the results of Autostep and provide new results on a test set of six new robot problems. Parameter setting for Autostep in these results: initial step-size value $\alpha_{i,0} = 0.1$ and $\lambda = 10^{-4}$.

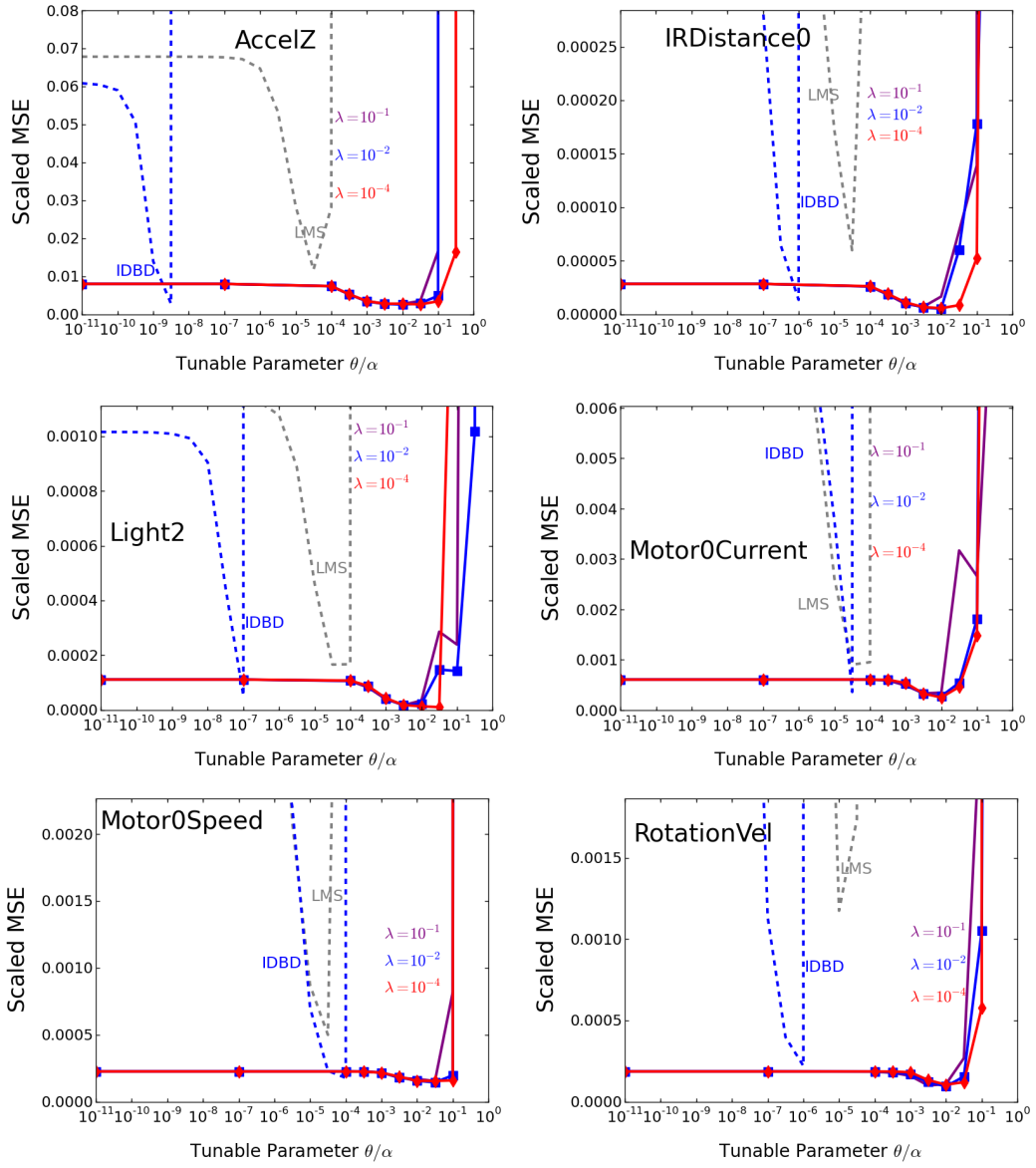

Figure 7.4: Performance vs. meta-step-size value for Autostep with three different settings of λ on six robot problems. Performance of IDBD and LMS is also given in dotted lines. Curves do not shift in an order of magnitude for different choices of λ.

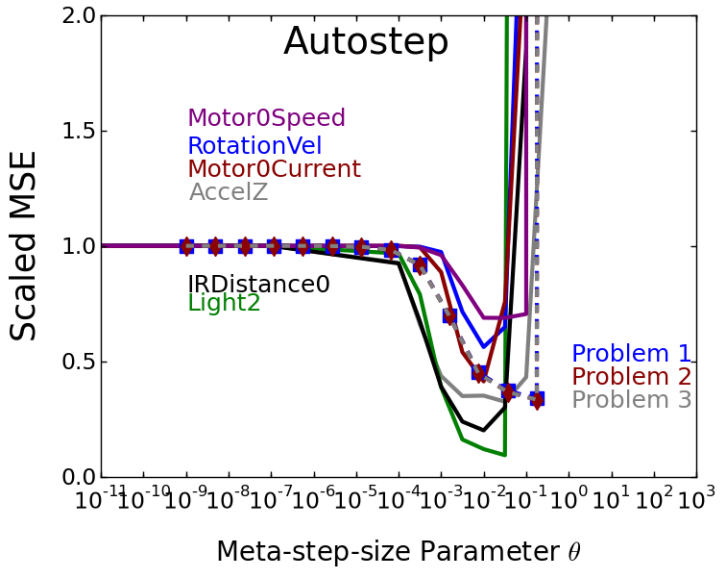

Figure 7.5: Summary of Autostep's performance with $\lambda = 10^{-4}$ on all three toy problems and six robot problems. For the meta-step-size value of 10^{-2}, Autostep achieves its near-best performance on all these problems.

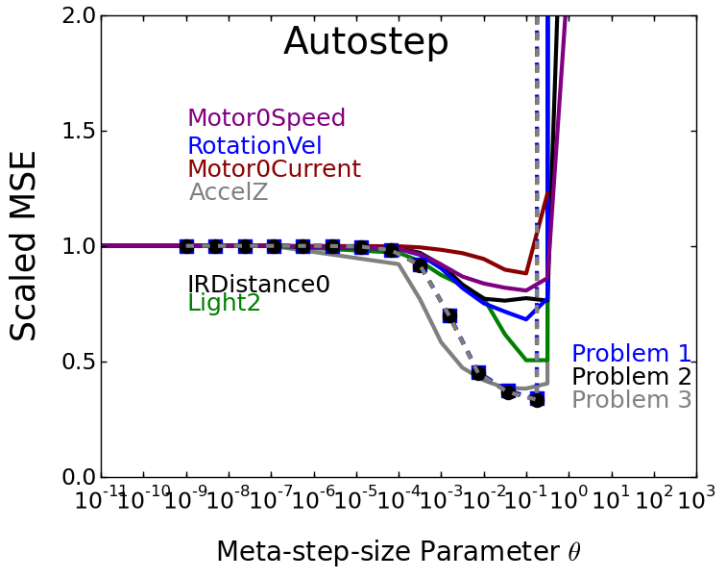

Figure 7.6: Validation of Autostep on six new robot problems. Here the inputs of the robot problems are unnormalized. For the meta-step-size value of 10^{-2} and $\lambda = 10^{-4}$, Autostep achieves its near-best performance on all these problems.

Figure 7.5 shows the performance curves of Autostep for different meta-step-size values on all of the test problems. Curves for the toy problems are shown in dotted lines. Near-best performance can be achieved with a meta-step-size value of 10^{-2} on all these problems. No other algorithms that we have experimented can adapt the step-size parameter effectively on all the test problems using the same parameter setting.

Figure 7.6 shows the results of Autostep on six new robot problems. In previous robot problems, the inputs were normalized. In these new robot problems, the inputs are unnormalized. Therefore, a learning algorithm faces two challenges. First, the problems are essentially nonstationary as is the previous robot problems. Second, the variance are now different for different inputs. Therefore, the performance surface can be elongated. In previous problems, due to normalization of inputs, the variance of the inputs were similar. The initial step-size value is 0.1 and $\lambda = 10^{-4}$.

Results on the new six robot problems show that Autostep again achieves near-best performance with a meta-step-size value of 10^{-2}. It means, our suggested choice of parameters also works well on this test set. This result validates that Autostep can effectively adapt step-size with the same setting of its parameters across widely different problems.

7.5 Discussion

In this chapter, we have summarized Autostep, our final algorithm of this work. Autostep contains four different parameters that can be set by the user: the meta-step-size parameter θ, the initial step-size value and the discounting factor λ of the running estimate. We have demonstrated that our algorithm can be used for adapting the step-size parameter effectively with the same setup of these parameters.

Autostep can achieve its near-best performance with meta-step-size values θ around 10^{-2} in all the problems that we have experimented including the training set containing three toy problems and six original robot problems and the test set including six new robot problems. Autostep is the least sensitive and most robust among all the existing step-size adaptation algorithms.

Autostep is also robust on the choice of the initial step-size values. A value of 10^{-2} or higher can be chosen for all the toy and robot problems and achieve near best performance. For IDBD, this choice is sensitive across problems and a wrong choice may even lead to divergence. For Autostep, we have experimented by setting initial step-size value as high as 100 and still haven't found Autostep diverging in any case. We have used 0.1 in most of our experiments and suggest to do so for an arbitrary problem.

For values of the discounting factor λ lower than 1, robustness of Autostep does not get affected. However, we have found that, smaller the value of the discounting factor, lesser is the sensitivity of Autostep to the meta-step-size parameter. We suggest a value of 10^{-4} for an arbitrary problem.

Autostep is a step-size adaptation algorithm that is the most robust and least sensitive to its parameters among all the algorithms that we have experimented. For an arbitrary problem, $\theta = 10^{-2}$, initial step-size value of 0.1 and $\lambda = 10^{-2}$ should be a good parameter setting to start with. Our results indicate that Autostep with this parameter setting can effectively adapt the step-size parameter in widely different problems. In extreme cases, when Autostep diverges for this setting, if at all, we suggest reducing the meta-step-size value, and then the discounting factor.

Chapter 8

Conclusions

In this thesis, we have introduced Autostep, a step-size adaptation algorithm, that is more robust and less sensitive to its tunable parameters than any other existing algorithms. We have verified this claim on two separate classes of problems consisting data from simulation and a real-world robotic platform. Autostep effectively adapts the step-size parameter on all these problems with the same setup of its tunable parameters.

By developing Autostep, we have achieved a significant milestone of our goals of step-size adaptation. Our goals of step-size adaptation include: superior performance than fixed step-size in non-stationary problems, linear time and memory complexity, adaptation of a vector step-size parameter and automatic adaptation without requiring any kind of manual tuning. All of our problems are potentially nonstationary and contain irrelevant inputs. Autostep adapts a vector step-size in LMS setting and performs significantly better than LMS in all of our experiments. It indicates that Autostep can effectively adapts the vector step-size parameter in such problems. Calculations of Autostep at each iteration only involves vector arithmetic and hence requires only linear time and memory complexity. Autostep adapts more automatically than any other existing algorithms. All the existing algorithms require manual tuning of one or more parameters. On the other hand, our results show that, in all of the problems, Autostep adapts the step size effectively with the same setup of its parameters. In our test problems, Autostep can be used in a completely automatic manner. For any arbitrary problem, it is also expected to be more automatic, requiring less tuning than any other algorithms. In that regard, Autostep

approaches all of our goals of step-size adaptation with a significant success.

Autostep comprises three new algorithmic techniques: meta-normalization, upper bounding the step-size update and pre-normalization. We have combined these ideas with the meta-descent technique of IDBD so that its meta-step-size parameter does not need manual tuning.

Although, they are combined with IDBD in this work, the techniques themselves are not specific to IDBD or supervised setting. All these techniques are gradient based. Our preliminary works indicate that our algorithm can be expressed in terms of sample gradients only. Therefore, if the sample gradients in a performance surface is analytically known in a problem, our algorithm can be extended to it. It would, of course, require re-derivation of the update rules of Autostep for the particular problem. However, the techniques of Autostep can be the starting point in avoiding manual tuning of free parameters.

How helpful are these techniques, when applied separately? Each of these techniques help step-size adaptation in becoming a bit more robust. These techniques, in fact, can also be considered separately in aiding automation of a system.

Autostep can be considered as another step toward the automation of step-size adaptation. It demonstrates that it is possible to design a step-size adaptation algorithm that can perform effectively in widely different problems without requiring tuning of its parameters. Autostep leads to new possibilities of developing algorithms in different problems that are less sensitive to their tunable parameters. In the following, we describe several natural directions for future works.

Extension to Other Step-Size Adaptation Algorithms: It can be possible to extend the ideas used in Autostep to other step-size adaptation algorithms. Autostep is developed based on IDBD and it applies a number of techniques to achieve an algorithm that is less sensitive to its meta-step-size parameter across problems. Nothing restricts these ideas from being extended to other steps-size adaptation algorithms, e.g., K1. The same ideas can be applied to achieve a problem independent meta-step-size in K1. Our preliminary results applying such techniques on K1 indicate that it might well be the case. An extensive work and experimentation is needed to develop a fully worked out

automatic step-size adaptation algorithm based on K1.

Finding theoretical Guarantees for Stability: As we demonstrated that Autostep with our suggested setup for parameters is stable on a number of problems, a theoretical guarantee of its stability should also exist. However, no such theory exists in the literature for any of the meta-descent algorithms we have presented here. Working on the theoretical analysis of these algorithms, including Autostep is a potential future direction.

Extension to Gradient-Based Reinforcement Learning Algorithms: Autostep techniques may also be extended to problems beyond supervised learning. New class of gradient-based reinforcement learning algorithms have recently been introduced for solving online reinforcement learning problems (Sutton et al., 2009). It introduces the possibility of solving a number of online robotic challenges, such as learning different tasks by using a single sequence of exploratory data. Currently, these algorithms use only scalar step-size parameters. Adaptation of a vector step-size parameter can enable these algorithms in performing more efficiently in large-scale, reinforcement learning tasks.

Extension of Autostep to these algorithms is a natural direction for a future work. A suitable range of meta-step-size values can be empirically found, which can then be used for many different online reinforcement learning problems without tuning it every time.

Extension to Non-Quadratic Optimization: A challenge for automatic step-size adaptation is to devise such an algorithm for problems with non-quadratic objective functions. In the linear supervised setting where Autostep is developed, the objective function is quadratic. Objective functions of many nonlinear systems are non-quadratic and many techniques developed for quadratic optimization does not readily apply to such systems. Extending Autostep to such problems and guaranteeing stability at every time-step is a great challenge and can be a potential future work.

Gradient-Based Adaptation of Arbitrary System Parameters: Adaptation of step-size parameter is only one instance of adapting tunable parameters. We have indicated in Chapter 1 that the step-size parameter is a form of bias. In general, any tunable parameter in a learning system forms a learning bias. It might be possible to adapt such tunable parameters of learning systems that currently require considerable amount of manual tuning. Some examples already exists. For example, Haykin (2001) adapted the exponential discounting factor γ of Recursive Least Squares filter using a gradient-based rule. Schaal and Atkeson (1997) adapted the parameters defining the shape of receptive fields in weighted regression using a rule similar to IDBD. Similarly, gradient-based approach can be applied to automatically adapt tunable parameters such as discounting factors, regularization parameters, etc., that exist in many systems. Extension of Autostep to these systems for adapting their arbitrary tunable parameters in a problem-independent manner can be a potential direction for future works.

We hypothesized that it is possible to adapt the step-size parameter more automatically than the existing algorithms. To establish this claim we developed Autostep and conducted a number of experiments with this algorithm on several different problems. We have established the statement by demonstrating that Autostep can effectively adapt the step-size parameter on all of our test problems using the same setting of its parameters. The establishment of this statement has opened up new possibilities in the applications of incremental learning.

Bibliography

Almeida, L. B., Langlois, T., Amaral, J., and Plakhov, A. (1998). Parameter adaptation in stochastic optimization. In Saad, D., editor, *On-Line Learning in Neural Networks*, pages 111–134. Cambridge University Press.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276.

Barkai, N., Seung, H. S., and Sompolinsky, H. (1995). Local and global convergence of on-line learning. *Physical Review Letters*, 75(7):1415–1418.

Bazaraa, M. S. and Shetty, C. M. (1979). *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons.

Bordes, A., Bottou, L., and Gallinari, P. (2009). SGD-QN: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754.

Bousson, K. (2007). Time-varying parameter estimation with application to trajectory tracking. *Aircraft Engineering and Aerospace Technology*, 79(4):406–413.

Bray, M., Koller-Meier, E., Müller, P., Gool, L. V., and Schraudolph, N. N. (2004). 3D hand tracking by rapid stochastic gradient descent using a skinning model. In *Proc. Intl. Conf. Artificial Neural Networks*, pages 59–68.

Diniz, P. S. (2002). *Adaptive Filtering: Algorithms and Practical Implementation*. Springer.

George, A. P. and Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65(1):167–198.

Goodwin, G. C. and Sin, K. S. (1984). *Adaptive Filtering Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall.

Haykin, S. (2001). *Adaptive Filter Theory (4th Edition)*. Prentice Hall.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45.

Ljung, L. (1998). *System Identification: Theory for the User (2nd Edition)*. Prentice Hall PTR.

Murata, N., Müller, K.-R., Ziehe, A., and ichi Amari, S. (1996). Adaptive on-line learning in changing environments. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *NIPS*, pages 599–605. MIT Press.

RLAI (2010). Critterbot homepage. `http://critterbot.rl-community.org/home`.

Schaal, S. and Atkeson, C. G. (1997). Receptive field weighted regression. Technical Report TR-H-209, ATR Human Information Processing Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gu, Kyoto 619-02, Japan.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. In *Proc. Intl. Conf. Artificial Neural Networks*, pages 569–574.

Schraudolph, N. N., Yu, J., and Aberdeen, D. (2006). Fast online policy gradient learning with SMD gain vector adaptation. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *NIPS*, volume 18, pages 1185–1192, Cambridge, MA. MIT Press.

Sompolinsky, H., Barkai, N., and Seung, H. S. (1995). On-line learning of dichotomies: Algorithms and learning curves. In *Neural networks: The statistical mechanics perspective. Proceedings of the CTP-PBSRI Joint Workshop on Theoretical Physics. Singapore: World Scientic.*, pages 105–130.

Sutton, R. S. (1981). Adaptation of learning rate parameters. Technical Report AFWAL-TR-81-1070, In: Goal Seeking Components for Adaptive Intelligence: An Initial Assessment, by A. G. Barto and R. S. Sutton. Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio 45433.

Sutton, R. S. (1992a). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proc. 10th National Conference on Artificial Intelligence*, pages 171–176.

Sutton, R. S. (1992b). Gain adaptation beats least squares. In *Proc. of the 7th Yale Workshop on Adaptive and Learning Systems*, pages 161–166.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In Danyluk, A. P., Bottou, L., and Littman, M. L., editors, *ICML*, volume 382 of *ACM International Conference Proceeding Series*, page 125. ACM.

Vijayakumar, S. and Schaal, S. (1998). Local adaptive subspace regression. *Neural Processing Letters*, 7(3):139–149.

Widrow, B. (1971). Adaptive filters. In Rudolf Emil Kalman, N. D., editor, *Aspects of Network and System Theory*, pages 563–586. Holt, Rinehart and Winston.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York. IRE.

Widrow, B., McCool, J., Larimore, M., and Johnson, J. (1976). Stationary and nonstationary learning characteristics of the LMS adaptive filter. *Proceedings of the IEEE*, 64(8):1151–1162.

Widrow, B. and Stearns, S. (1985). *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In Fawcett, T. and Mishra, N., editors, *ICML*, pages 928–936. AAAI Press.