Online Learning in Vowpal Wabbit

Thomas Øvestad

Master of Science in Physics and Mathematics
Submission date: December 2015
Supervisor: Håvard Rue, MATH
Co-supervisor: Thiago Martins, Yahoo Technologies Norway

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Statement

The purpose of this thesis is to study techniques that are useful in large scale data analysis, specifically in an online learning setting, by focusing on the methods implemented in the statistical software Vowpal Wabbit. This consists of providing a theoretical introduction to extensions of stochastic gradient descent and conducting experiments to document their performance.

Assignment given: September 3, 2015
Ammended: November 25, 2015
Supervisor: Håvard Rue, Department of Mathematical Sciences, NTNU
Co-supervisor: Thiago Martins, Yahoo!

Preface

This thesis concludes the candidate's MSc in Applied Physics and Mathematics with a specialisation in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU). The thesis is written at the Department of Mathematical Sciences under the supervision of Professor Håvard Rue.

The subject of the assignment was motivated by my co-supervisor Thiago Martins, data scientist at Yahoo!, the global internet company. It has been both thrilling and challenging to evaluate and develop online learning methods with the constraints of the Big Data era.

I would like to thank Håvard and Thiago for all the discussions, the words of encouragement and for reading through drafts of the thesis. The collaboration has been extremely rewarding and inspiring.

Thomas Øvestad
Trondheim, December 31, 2015

Abstract

Online learning methods for sequentially arriving data are growing in popularity. Alternative batch learning methods scale poorly and have memory constraints. The scope of this thesis is to study online learning methods that are based on stochastic gradient descent, or SGD, and are implemented in Vowpal Wabbit, an increasingly popular online learning software. The literature and experiments on these methods reveal that, despite scaling well, they are only designed for data originating from stationary models. This is an important weakness, as the data for which these models are necessary will often be nonstationary in nature. We propose a new framework that builds on the SGD algorithm. For every incoming example Parallellised SGD, or PSGD, runs alternative SGD-learners with different learning rates in parallell to a chosen SGD learner. The alternative learners help tune the chosen learning rate by sequentially comparing the errors of the learners. This provides a scalable framework as the gradient still only needs to be computed once per example, and the added computational cost of the alternative learners can be diminished through efficient parallelisation. Experiments on a proof-of-concept implementation demonstrate that PSGD is superior to Vowpal Wabbit's SGD-based implementations in nonstationary settings. However, further work is needed to improve the adaptiveness of the method to a wider range of nonstationary behaviour. Sustained research on this framework shows great promise to yield a class of adaptive learners that automatically handle nonstationary data and can be subject to large scale implementations in online learning softwares such as Vowpal Wabbit.

Sammendrag

Online learning-metoder for kontinuerlige datastrømmer øker i popularitet. Alternative batch learning-metoder skalerer dårlig og har minnebegrensninger. Formålet med denne masteroppgaven er å studere online learning-metoder som er basert på stochastic gradient descent, SGD, og er implementert i Vowpal Wabbit, et populært online learning-program. Litteraturstudiet og egne eksperimenter viser at metodene er tilpasset stasjonære datasett. Dette er en vesentlig svakhet siden skalerbarheten til disse metodene ofte er nødvendig for å analysere datasett preget av ikke-stasjonæritet. Vi foreslår et nytt rammeverk som bygger på SGD-metoden. For hvert nye datapunkt gjennomfører Parallellisert SGD, eller PSGD, alternative SGD-steg med forskjellige steglengder parallellt til SGD-steget med den valgte steglengden. Disse alternative stegene kan brukes til å adaptivt justere steglengden ved å sekvensielt sammenligne feilene. Dette rammeverket er skalerbart siden man kun trenger å evaluere gradienten en gang for hvert datapunkt og kostnaden ved å gjennomføre flere SGD-steg per datapunkt kan begrenses ved å parallellisere metoden. Eksperimenter på en prototypeimplementasjon viser at PSGD gir mer tilfredstillende resultat enn Vowpal Wabbit sine SGD-baserte metoder for ikke-stasjonære datasett. Rammeverket krever imidlertid videreutvikling for å ytterligere forbedre PSGDs adaptivitet for et videre spenn av ikke-stasjonæritet. PSGD viser et stort potensial for å danne en klasse av sekvensielle adaptive metoder for ikke-stasjonære datasett som kan bli implementert i online learning-program som Vowpal Wabbit.

Contents

Problem Statement **i**

Preface **iii**

Abstract **v**

Contents **viii**

1 Introduction **1**

2 Batch learning **3**
2.1 Regression - An Optimisation Problem 3
2.1.1 Loss Function Optimality 4
2.1.2 Likelihood Optimality 7
2.2 Numerical Batch Optimisation 7

3 Online Learning in Vowpal Wabbit **11**
3.1 Stochastic Gradient Descent 11
3.1.1 Justification of SGD 13
3.1.2 How Asymptotic Behaviour Depends on Step-Size . . . 14
3.2 Importance Weights 15
3.3 AdaGrad: Treat Massive, Rare and Sparse Data 18
3.4 Online Normalisation 21
3.5 Truncated Gradient: Online Sparsification 23
3.6 Model Assessment for Online Learning 26
3.6.1 Foundational Concepts for Batch Learning 26
3.6.2 Cross Validation 27
3.6.3 Progressive Cross Validation 28

4 Design of Experiments **30**
4.1 Models 30
4.1.1 Simple Linear Model 31
4.1.2 Time Series Models 32

4.1.3 Implementation of Regression 32
4.2 Experiments 33
4.2.1 Static Setting 33
4.2.2 Dynamic Setting 37
4.3 Model Assessment 40

5 Results 42
5.1 Static Setting 42
5.1.1 Effect of Signal to Noise Ratio 42
5.1.2 Effect of More Parameters 50
5.2 Dynamic Setting 58
5.2.1 AdaGrad on Time Series Models 58
5.2.2 Fixed Learning Rates on Time Series Models 72

6 PSGD 80
6.1 Motivation 80
6.2 Method 81
6.3 Design of Experiments 84
6.4 Results 87
6.4.1 AdaGrad Comparison 87
6.4.2 Fixed Learning Rate Comparison 93
6.4.3 Nonstationary Data from Multiple Models 98
6.5 Conclusion and Further Work 105

7 Summary 108

References 110

Appendices 113
A Implementation of Experiments 113
B Derivations for Linear Model Experiments 114
C Additional Figures 116
C.1 Error Plots for Exp. D4-D6 116
C.2 Error Plots for Exp. M1-M3 120

1 Introduction

In the past decades computational processing power, spurred on by Moore's law [18], has seen rapid performance increases. At the same time, personal computers, mobile phones and other devices are becoming more abundant and connect ever more people to the internet. The result is an immense growth in digital data traffic and storage [8]. Digital data sets, for example from large internet sites, are often so large that common statistical data analysis methods break down. One reason for this is that these models, as products of their time, were developed with smaller data sets in mind. The answer to the abundance of huge data sets has been a growing focus on developing computationally efficient methods, algorithms and numerical approximations to replace older computationally infeasible methods. This emerging field is often termed Big Data, an intersection between the fields of statistics and computer science [24].

This thesis will consider online settings where data sets are continuously increasing and have millions of rows, thereby requiring the treatment of statistical methods for online large-scale data analysis. Currently, there is a strong research interest in two different approaches to analysing ever growing data sets. One approach is to develop methods that *parallelise* well over multiple processor cores [7]. Another is to increase the speed and decrease the memory requirements of existing methods. This last approach relies on approximations and therefore present a trade-off between computational efficiency and accuracy, but can be easier to apply for analysing continuous data streams. This thesis will study some of these fast approximative methods, that are implemented in the statistics software, Vowpal Wabbit.

Vowpal Wabbit (`vw`) is an open-source software with extremely fast implementations of multiple learning algorithms for several loss functions using different optimisation algorithms. The work is sponsored by Microsoft Research and (previously) Yahoo! Research. The main reason why the implementations are so fast is because they are based on Stochastic Gradient Descent, or SGD, which do not have the same memory and performance constraints as traditional batch learning algorithms[1][2].

This thesis can be divided into three parts. The first constitutes a literature study that is relevant to understand SGD-based online learning in vw. In order to motivate and contextualise online learning the thesis starts by briefly introducing Batch Learning in section 2, before treating online learning in vw in section 3. The second part of the thesis is experimental and seeks to shed light on selected aspects of the discussed online learning methods. The design of the experiments is treated in section 4 and the results are presented in section 5. The main conclusion from this part is that the SGD-based online learning methods implemented in vw, are not designed to fit models to nonstationary data in an online setting. In section 6, the last part of this thesis, we propose a new online learning method called Parallelised Stochastic Gradient Descent, or PSGD, to address this problem. The section includes a motivation and explanation of the method, experimental results demonstrating PSGD's strengths compared to the SGD-based methods implemented in vw and a list of aspects that require improvements and further work.

Further research on this framework has the potential to yield a class of adaptive learners that can automatically handle nonstationary data and be subject to large scale implementations in online learning software such as Vowpal Wabbit.

[1]Batch learning algorithms are statistical methods that load all observational data into memory. Online learning algorithms only load one or a few observations into memory at any given time, and can be run on a continuous data stream.

[2]See `https://github.com/JohnLangford/vowpal_wabbit/wiki` for a more thorough overview and introduction to the software.

2 Batch learning

The focus in this thesis is on solving regression problems in realistic scenarios where the available data exceeds the RAM[3] of a normal computer and arrives continuously. Solving regression problems requires maximising a likelihood function or minimising a risk function. Several optimisation methods can do this. This section introduces some batch learning procedures for solving regression problems that put the online learning algorithms implemented in Vowpal Wabbit, discussed in section 3, in perspective. The optimisation methods used for batch learning are key, because they are modified to yield the online learning methods in Vowpal Wabbit.

2.1 Regression - An Optimisation Problem

One of the important problems in statistics is the study of the mathematical relationship between different types of observations. Customarily one distinguishes between observations from *explanatory* variables, also called predictors, and *response* variables, where the latter are functions of, or explained by, the former [2]. In this paradigm one can represent the relationship between responses and predictors as

$$y_i = f(\boldsymbol{x}_i|\boldsymbol{\beta}) + \epsilon_i, \quad i = 1, ..., n. \tag{2.1}$$

where $\boldsymbol{\beta}$ is the vector of regression coefficients, y_i is the i^{th} response, $\boldsymbol{x_i}$ is the i^{th} predictor vector, ϵ_i is the disturbance term or noise related to the i^{th} observation and n is the total number of observations. The signal of the model is here denoted by f and is separated from the noise ϵ. In this thesis only linear models will be used for model fitting, although some time series models will also be used for simulating data. For linear models the signal of the model, f, is termed the linear predictor, and can

[3]Random access memory.

be written on the following form

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = \boldsymbol{x_i}^\top \boldsymbol{\beta} + \epsilon_i \quad i = 1, ..., n \tag{2.2}$$

Generalising to n observations we have the compact relationship

$$\boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{2.3}$$

It is important to note that on the right-hand side of equation (2.1) the disturbances, ϵ_i, are stochastic terms. In Bayesian settings this also includes $\boldsymbol{\beta}$. This implies that y_i is stochastic.

The goal of a regression analysis is to estimate the regression coefficients $\widehat{\beta}_j$ that yield the optimal mathematical relationship between predictors $\boldsymbol{x}_i$ and responses y_i, for a given model. From these coefficient estimates one obtains a model that can be used to estimate the signal given observed predictors $\boldsymbol{x}$. These estimates, $\widehat{y}$, are computed as follows

$$\widehat{\boldsymbol{y}} = \boldsymbol{x}^\top \widehat{\boldsymbol{\beta}} \tag{2.4}$$

How we estimate $\widehat{\beta}_j$ naturally depends on what we mean by *optimal*.

2.1.1 Loss Function Optimality

Historically, the first approach used to solve this problem was to introduce a loss function that quantifies how well the model fits the data.

$$L = L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) \tag{2.5}$$

The loss function, L, in equation (2.5) computes a measure of the discrepancy of the responses, $\boldsymbol{y}$ and the estimates, $\widehat{\boldsymbol{y}}$. Note that the loss function is a linear combination of a pointwise loss function, $L_{\text{point}}(\boldsymbol{y}, \widehat{\boldsymbol{y}})$ and a regularization function, $L_{\text{reg}}(\boldsymbol{y}, \widehat{\boldsymbol{y}})$,

$$L = L_{\text{point}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) + L_{\text{reg}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}). \tag{2.6}$$

Common examples of pointwise loss functions are [5]

$$L_{\text{point}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \begin{cases} I(\boldsymbol{y} \neq \widehat{\boldsymbol{y}}), & \text{0-1 loss} \\ |\boldsymbol{y} - \widehat{\boldsymbol{y}}|, & \text{absolute error loss} \\ (\boldsymbol{y} - \widehat{\boldsymbol{y}})^2, & \text{squared error loss} \end{cases} \tag{2.7}$$

Note that what is called Least Squares Regression[4] has a squared error loss function and that the indicator function I is defined as

$$I(\boldsymbol{y} \neq \widehat{\boldsymbol{y}}) = \begin{cases} 1, & \text{if } \boldsymbol{y} \neq \widehat{\boldsymbol{y}} \\ 0, & \text{if } \boldsymbol{y} = \widehat{\boldsymbol{y}}. \end{cases} \tag{2.8}$$

Some common regularization functions are

$$L_{\text{reg}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \begin{cases} \lambda_1 ||\boldsymbol{\beta}||_1, & \text{L1-regularization} \\ \lambda_2 ||\boldsymbol{\beta}||_2, & \text{L2-regularization} \\ \lambda_1 ||\boldsymbol{\beta}||_1 + \lambda_2 ||\boldsymbol{\beta}||_2, & \text{Elastic Net [27]} \end{cases} \tag{2.9}$$

where $\lambda_1, \lambda_2 > 0$ are real regularization constants. Combining squared error loss with L1 and L2 regularization yield, respectively, Lasso and Ridge regression [13].

In order to solve the regression coefficient estimation problem and perform the estimation in equation (2.4), one needs a measure of the overall loss of the regression model on a set of test observations, not just a pointwise loss (and regularization). The remedy is what is termed the risk function, $R(\boldsymbol{\theta})$, a weighted average of the loss of the considered test observations[5]

$$R(\boldsymbol{\beta}) = E_f\big[L(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\beta})\big] = \int L(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\beta}) f(\boldsymbol{X}, \boldsymbol{Y}) d\mathsf{x} d\mathsf{y} \tag{2.10}$$

[4]The famous mathematicians Legendre and Gauss were the first to publish work on Least Squares regression, which they used to fit planetary orbits [2].

[5]Two comments on the notation of equation (2.10).

1. $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{x})) = L(\boldsymbol{x}, \boldsymbol{y})$

2. In order to calculate the expected value one needs to consider the observations $\boldsymbol{x}, \boldsymbol{y}$ as random stochastic variables $\boldsymbol{X}, \boldsymbol{Y}$.

where $f(\boldsymbol{X}, \boldsymbol{Y})$ is the joint probability distribution of random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$. The higher the value of a risk function, the greater the mismatch between responses and estimates based on the model. Hence the risk function represented by equation (2.10) represents how well the regression coefficients in equation (2.4) are estimated. One can therefore approach the regression coefficient estimation problem by minimizing the risk function with respect to the coefficients

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\beta} R(\boldsymbol{\beta}). \tag{2.11}$$

In practice, the joint distribution function, $f(\boldsymbol{x}, \boldsymbol{y})$ is not known and needs to be approximated by an empirical distribution $\widehat{f}(\boldsymbol{x}, \boldsymbol{y})$ such as [19]

$$\widehat{f}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \delta(x - x_i, y - y_i) \tag{2.12}$$

where $\delta(x, y)$ is the two-dimensional Dirac measure defined by

$$\delta(x, y) = \begin{cases} 0, & (x, y) \neq (0, 0) \\ 1, & (x, y) = (0, 0) \end{cases} \tag{2.13}$$

yielding the empirical risk function

$$\widehat{R}(\boldsymbol{\beta}) = E_{\widehat{f}}\big[L(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\beta})\big] = \int L(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\beta}) \widehat{f}(\boldsymbol{X}, \boldsymbol{Y}) d\mathsf{x} d\mathsf{y} \tag{2.14}$$

which is more commonly estimated from

$$\widehat{R}(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i) \tag{2.15}$$

and is subsequently minimised to obtain an empirical estimate of the regression coefficients

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\beta} \widehat{R}(\boldsymbol{\beta}). \tag{2.16}$$

2.1.2 Likelihood Optimality

Almost a century later, Fischer, presented a different solution to the parameter estimation problem [2]. Since the responses are stochastic, an approach to estimating the coefficients is to determine which regression coefficients are the most likely given the observed data.

The problem is typically framed as follows. Given n responses, $\boldsymbol{y}$, observed from some distribution, $f(\boldsymbol{y}|\boldsymbol{\beta})$, with unknown parameters, $\boldsymbol{\beta}$, what are the most likely parameters of the distribution? The key in answering this question is to introduce the likelihood function[6]

$$\mathcal{L}(\boldsymbol{\beta}|\boldsymbol{y}) \coloneqq f(\boldsymbol{y}|\boldsymbol{\beta}). \tag{2.17}$$

The likelihood function $\mathcal{L}$ quantifies how likely it is that the observations, $\boldsymbol{y}$, originate from a distribution with parameters, $\boldsymbol{\beta}$. Thus it is clear that the aforementioned question is answered by maximising the likelihood in equation (2.17) with respect to the regression coefficients

$$\widehat{\boldsymbol{\beta}} = \arg\max_{\beta} \mathcal{L}(\boldsymbol{\beta}|\boldsymbol{y}). \tag{2.18}$$

Note that $\widehat{\boldsymbol{\beta}}$ is called the maximum likelihood estimator in likelihood theory.

2.2 Numerical Batch Optimisation

As shown in section 2.1, regression reduces to optimisation problems exemplified by equations (2.11) and (2.18). These minimisation and maximisation problems can sometimes be solved analytically. This consists of finding the global extrema of the objective functions, R or $\mathcal{L}$, over

[6]It is important to note that, though not immediately apparent due to challenges with conventional notation, the likelihood function is, in a regression setting, conditioned on the observed predictors, $\boldsymbol{x}$.

the parameter space or domain. The difficulty of this depends on the properties of the objective functions, the number of observations and the dimensionality of the parameter space. The applications considered in this thesis are designed for high dimensional parameter spaces and for a lot of data, therefore requiring numerical optimisation. In this section we consider general numerical batch optimisation methods that are not developed specifically for risk minimisation or likelihood maximisation, but which could be used for both. This section discusses unconstrained optimisation in particular, and introduces line search methods such as the gradient descent algorithm. Gradient descent is discussed because it is adapted to an online learning method that is central to section 3.

In unconstrained optimisation one considers problems of the form

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^n} f(\boldsymbol{\beta}) \tag{2.19}$$

where f is the objective function with parameter vector $\boldsymbol{\beta}$. The minimization can be carried out by similar or identical algorithms needed to solve the regression optimisation problems in equations (2.11) or (2.16) and (2.18) depending on the domains, likelihood and risk functions at hand.

Line search methods. For every iterate k, these methods compute a search direction, $\boldsymbol{p}_k$ from a point, $\boldsymbol{\beta}_k$ on the objective function[7]. The algorithm then computes the step length, α_k, i.e. how far along the search direction to move, hence computing the next point, $\boldsymbol{\beta}_{k+1}$

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \alpha_k \boldsymbol{p}_k \tag{2.20}$$

The search direction is of the form

$$\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{\beta}_k) \tag{2.21}$$

[7]This thesis only considers convex objective functions, f, satisfying $f\left(tx_1 + (1-t)x_2\right) \leq tf(x_1) + (1-t)f(x_2) \; \forall \; x_1, x_2 \in X, \; t \in [0,1]$, where X is the domain of f.

because then we have

$$\boldsymbol{p}_k^\top \nabla f(\boldsymbol{\beta}_k) = -\nabla f(\boldsymbol{\beta}_k)^\top \boldsymbol{B}_k \nabla f(\boldsymbol{\beta}_k) \leq 0, \tag{2.22}$$

where $\boldsymbol{B}_k$ is the matrix of the objective function's second derivatives, which is positive semi-definite for convex function f. The search direction for each iterate is subsequently chosen by finding the local or, if possible, the global minimum of

$$\phi(\alpha) = f(\boldsymbol{\beta}_k + \alpha \boldsymbol{p}_k), \quad \alpha > 0. \tag{2.23}$$

In practice searching for a global minimum might be computationally infeasible and finding each local minimum might reduce the convergence rate. To remedy this one can use inexact line-search methods to find step lengths that satisfy the Wolfe or Goldstein conditions [20].

A problem with Newton methods, which have $\boldsymbol{B}_k$ equalling an exact or approximate matrix of second derivatives at a point is that these computations can be expensive. A solution to this is to simplify the line search algorithm by using $\boldsymbol{B}_k = \boldsymbol{I}$, the identity matrix. This yields what is known as the *Steepest Descent* or the *Gradient Descent* method, first introduced by Cauchy (1847).

Line search on risk functions. The aforementioned methods can be used to minimise the objective function we termed the risk and empirical risk functions in equations (2.11) and (2.16), respectively. For instance, applying gradient descent on the empirical risk function yields[8]

$$\begin{aligned}\boldsymbol{\beta}_{k+1} &= \boldsymbol{\beta}_k - \alpha_k \nabla \widehat{R}(\boldsymbol{\beta}_k) \\ &= \boldsymbol{\beta}_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\beta}} L\big(\boldsymbol{y}_i, \widehat{\boldsymbol{y}}_i(\boldsymbol{x}_i) | \boldsymbol{\beta}_k\big)\end{aligned} \tag{2.24}$$

The machine learning literature refers to the step size, α, as the learning rate and denotes learning algorithms of the form of equation (2.24)

[8]Note that we are differentiating with respect to $\boldsymbol{\beta}$ in equation (2.24).

as *batch* learning algorithms. This is because for each iteration of the parameter estimate, $\boldsymbol{\beta}_k$, the computer needs to go through each of the n observations at hand. For vast data sets this can pose a significant constraint on the run-time of the algorithm.

3 Online Learning in Vowpal Wabbit

Most of the literature on applying stochastic optimisation to regression problems studies the problem of minimizing a risk function, and will also be the focus of the rest of this thesis. It is important to bear in mind that one can also use stochastic optimisation methods to maximise a likelihood function.

This section presents a mathematical foundation of essential[9] aspects of the online learning methods implemented in Vowpal Wabbit, some of which will be the subject of the experimental part of the thesis, in sections 4 and 5.

3.1 Stochastic Gradient Descent

When faced with an unusually large data set, either because the data is continuously arriving, or because its size is too large to be handled by the random access memory of ordinary computers, a possibility is to approach the regression coefficient optimisation problem by further simplifying the gradient descent method in equation (2.24). A method that has sparked interest in recent years is named *Stochastic Gradient Descent*, which only uses the gradient at a randomly[10] chosen example, $\boldsymbol{z}_k = (\boldsymbol{x}_k, \boldsymbol{y}_k)$, from the complete set of observations $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ for each k^{th} example,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_i, \widehat{\boldsymbol{z}}_i | \boldsymbol{\beta}_k). \tag{3.1}$$

One can also use a stochastic version of a Newton method

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha_k \boldsymbol{B}_k^{-1} \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_i, \widehat{\boldsymbol{z}}_i | \boldsymbol{\beta}_k). \tag{3.2}$$

[9] For more information see `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

[10] In an online setting where data is continuously arriving, SGD needs to process each incoming data point in its order of arrival. Therefore no randomisation is performed in an online setting.

though this version is seldom used in practice since the conditions that warrant this extent of simplification make it difficult to justify an iterative computation of the inverse of the hessian, $\boldsymbol{B}^{-1}$. AdaGrad, an extension of SGD that resembles the Newton-like version in equation (3.2) is implemented in Vowpal Wabbit and will be discussed in more detail in section 3.3.

Note that the sequence of coefficient estimates, $\{\boldsymbol{\beta}_k\}$, for positive integers, k, is (approximately) optimising the risk function without recalling the visited observations, $\boldsymbol{z}_l$, for $l < k$. Additionally, since SGD only processes one example, $\boldsymbol{z}_k$, at a time, the method can handle a continuous stream of data arriving. It is for this reason that these types of learning algorithms are termed *online learners*. One can think of each iteration as sampling random examples from the *grand truth* distribution[11] in order to minimise the risk function [4].

It is evident that the stochastic gradient descent method in equation (3.1) is a significant simplification of the gradient descent method of equation (2.24), which takes all examples into account. In fact, the larger the data set, the larger the simplification of the online learner. A possible remedy [19] is to evaluate $m < n$ observations, $\{(\boldsymbol{x}_k^i, \boldsymbol{y}_k^i)\}_{i=1}^m$, for each iterate, k,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \frac{\alpha_k}{n} \boldsymbol{B}_k^{-1} \sum_{i=1}^{m} \nabla_{\boldsymbol{\beta}} L\big(\boldsymbol{y}_k^i, \widehat{\boldsymbol{y}}_k^i(\boldsymbol{x}_k^i) | \boldsymbol{\beta}_k\big). \tag{3.3}$$

This method is a *mini-batch* version of SGD, where m is called the mini-batch size. An overview of other variants of stochastic gradient algorithms are given by Bottou (2010).

[11]The grand truth distribution is the unknown distribution of random variables $(\boldsymbol{X}, \boldsymbol{Y})$ that the regression is trying to model.

3.1.1 Justification of SGD

For time-independent, $\boldsymbol{\beta}$, one can justify the asymptotic convergence of SGD described in equation (3.1) by considering the expected value of the parameter estimates of the more general stochastic Newton method in equation (3.2).

First let us consider the expected risk, $E_Z(\cdot)$, with respect to random observations $Z = (X, Y)$

$$E_Z\big[R(\boldsymbol{\beta}_{k+1})\big] = E_Z\Big[R\big(\boldsymbol{\beta}_k - \alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_i, \widehat{\boldsymbol{z}}_i | \boldsymbol{\beta}_k)\big)\Big]. \tag{3.4}$$

In order to evaluate the expectation we use equation (A.7)[12] in the proof of Lemma 2 in Murata (1998). In the notation adopted by this paper, the result is

$$E_Z\Big[f\big(\boldsymbol{\beta} - \alpha \boldsymbol{B}^{-1} \nabla L(\boldsymbol{z}|\boldsymbol{\beta})\big)\Big] = f(\boldsymbol{\beta}) - \alpha \nabla f(\boldsymbol{\beta})^\top \boldsymbol{B}^{-1} \nabla R(\boldsymbol{\beta}) + O(\alpha^2). \tag{3.5}$$

where $f(\cdot)$ is some sufficiently smooth function. By setting $f(\cdot) = R(\cdot)$ and $\boldsymbol{B} = \boldsymbol{I}$ one can insert equation (3.5) into equation (3.4) and obtain

$$E_Z\big[R(\boldsymbol{\beta}_{k+1})\big] = R(\boldsymbol{\beta}_k) - \alpha \nabla R(\boldsymbol{\beta}_k)^\top \boldsymbol{B}_k^{-1} \nabla R(\boldsymbol{\beta}_k) + O(\alpha^2). \tag{3.6}$$

This can be rearranged to yield a result implying that the expected risk of the $(k+1)$th parameter estimate is smaller than the risk of the kth parameter estimate

$$E_Z\big[R(\boldsymbol{\beta}_{k+1})\big] - R(\boldsymbol{\beta}_k) = -\alpha \nabla R(\boldsymbol{\beta}_k)^\top \boldsymbol{B}_k^{-1} \nabla R(\boldsymbol{\beta}_k) + O(\alpha^2) < 0 \tag{3.7}$$

since $\boldsymbol{B}_k$ is a positive definite matrix. The result justifies the stochastic version of the Newton method, as well as the Stochastic Gradient Descent method where $\boldsymbol{B}_k = \boldsymbol{I}$. The assumptions required to obtain the result in equation (3.7) are that $\boldsymbol{B_k}$ is fixed and that α_k satisfies the conditions of equation (3.8).

[12]The result is obtained by Taylor expansion about the true parameter $\boldsymbol{\beta}$.

3.1.2 How Asymptotic Behaviour Depends on Step-Size

Murata (1998) builds on the convergence criteria for the step sizes in Robbins' and Monro's theory of stochastic approximations [21]

$$\sum_{i=1}^{\infty} \alpha_i = \infty, \quad \sum_{i=1}^{\infty} \alpha_i^2 = \infty \tag{3.8}$$

and proves several asymptotic convergence results[13] for different step sizes in both online and batch learning systems. Note that only time-independent parameters, $\boldsymbol{\beta}$, are considered in the following.

For fixed step sizes in an online learning system, there is a bias of the order $O(\alpha)$ caused by the ever fluctuating estimates

$$\mathrm{E}\left[R(\boldsymbol{\beta}_k)\right] = R(\boldsymbol{\beta}) + O(\alpha) + g(k) \tag{3.9}$$

where $g(k)$ is the decaying time-dependent[14] component of the expected risk. It is clear that fixed step sizes do not satisfy the convergence criteria in equation (3.8), justifying the bias in the expected risk.

Annealed, or time-decaying, step sizes can satisfy the convergence criteria in equation (3.8). Step sizes of the form $\alpha_k = O(1/k)$ satisfy these criteria. Assuming that the covariance matrix of the estimates are of a certain form[15] and we thereby deal with optimally annealed learning with a step size of $O(1/k)$, we have the following asymptotic expected value result

$$\mathrm{E}\left[R(\boldsymbol{\beta})\right] = R(\boldsymbol{\beta}) + O(1/k). \tag{3.10}$$

Note that Murata (1998) shows that optimally annealed online learning,

[13]Though all the mentioned results are accurate, some of the terms are condensed into big-O notation to make the section fit for purpose. The interested reader may consult the references for more detailed results.

[14]In the sense that time-dependence means step-size dependence.

[15]See Corollary 3 section 2.2 in Murata (1998).

in equation (3.10), is asymptotically as efficient as batch learning in terms of the generalization error[16].

3.2 Importance Weights

The notion of *importance weights* for the stochastic gradient descent algorithm, represented by equation (3.1), has not yet been discussed. Importance weights quantify the relative importance of one data point[17] compared to another. I.e. if a data point has an importance weight of h then this is equivalent to having the data point featuring h times in the data set. Thus it is natural to desire that online learning algorithms satisfy *importance invariance*.

Importance invariance property. *For all importance weights, h, the update of the learning algorithm is equivalent to n updates with importance weight h/n.*

The most intuitive implementation is to multiply the gradient in the update rule by h

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - h\alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_i, \widehat{\boldsymbol{z}}_i | \boldsymbol{\beta}_k). \tag{3.11}$$

If this is to satisfy the importance invariance property then we require that updating the estimates $n > 0$ times, using step-size h/n, should yield the

[16]For more information about the generalization error, see section 3.6.1

[17]In the machine learning literature one refers to data points as *examples*.

same coefficient estimate. This n-step procedure can be represented by

$$\begin{aligned}
\boldsymbol{\beta}_{k,1} &= \boldsymbol{\beta}_k + \frac{h}{n}\alpha_k \nabla L(\boldsymbol{y}_k, \widehat{\boldsymbol{y}}_k) \\
\boldsymbol{\beta}_{k,2} &= \boldsymbol{\beta}_{k,1} + \frac{h}{n}\alpha_k \nabla L(\boldsymbol{y}_k, \widehat{\boldsymbol{y}}_{k,1}) \\
&\vdots \\
\boldsymbol{\beta}_{k,n} &= \boldsymbol{\beta}_{k,n-1} + \frac{h}{n}\alpha_k \nabla L(\boldsymbol{y}_k, \widehat{\boldsymbol{y}}_{k,n-1})
\end{aligned} \tag{3.12}$$

where the n-step update, $\boldsymbol{\beta}_{k,n}$ corresponds to one update with a full importance weight, $\boldsymbol{\beta}_{k+1}$ and

$$\widehat{\boldsymbol{y}}_{k,i} = \boldsymbol{x}^\top \widehat{\boldsymbol{\beta}}_{k,i}\ , \quad i = 1, ..., n. \tag{3.13}$$

However, for $h \neq 1$

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_{k,n} \tag{3.14}$$

is only satisfied for linear loss functions, L. This is an infeasible restriction.

John Langford, Vowpal Wabbit's main developer, co-authored a paper addressing this issue and implemented the method in the software [15]. The first step is rewriting the n-step as a recursive relation.

Lemma 3.1. *Let $h \in \mathbb{N}$. Presenting data point ($\boldsymbol{x}, \boldsymbol{y}$) h times in a row is equivalent to the update*

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + s(h)\boldsymbol{x} \tag{3.15}$$

where the scaling factor s(h) is defined by the recursive relation.

$$\begin{aligned}
s(h+1) &= s(h) + \alpha_k \nabla L|_{\boldsymbol{\beta}=\boldsymbol{\beta}_k + s(h)\boldsymbol{x}} \\
s(0) &= 0
\end{aligned} \tag{3.16}$$

Note that Lemma 3.1 can be proved by induction based on equations (3.12). Its importance is due to the fact that solving the recursive relation for $s(h)$ is equivalent to finding an update rule that preserves the invariance property. Karampatziakis and Langford (2011) demonstrate that one can derive closed-form expressions for $s(h)$ for certain loss functions, such as the squared error loss. An arguably more important result is their theorem[18] representing the scaling factor by an ordinary differential equation, ODE.

Theorem 3.2. *The limit of the stochastic gradient descent process of the j^{th} component of the coefficient vector $\boldsymbol{\beta}$, as the step-size becomes infinitesimal for a data point with a positive importance weight, $h \in \mathbb{R}$, is equal to the update*

$$\beta_{k+1,j} = \beta_{k,j} - s(h)x_j \tag{3.17}$$

where the scaling factor $s(h)$ satisfies the ordinary differential equation

$$s'(h) = \alpha_k \frac{\partial L}{\partial \beta_j}|_{\beta_j=\beta_{k+1,j}}, \quad s(0) = 0 \tag{3.18}$$

for all dimensions $j = 1, ..., d$ of the coefficient vector.

This can also be used to find some closed-form expressions for $s(h)$, by plugging loss functions into equation (3.18) and solving the ODE by separation of variables.

Karampatziakis and Langford (2011) also prove that the ODE in equation (3.18) in Theorem 3.2 preserves the importance invariance property and can be adapted to annealed step sizes.

[18]The proof starts by expressing $s(h + \epsilon)$ in terms of $s(h)$ and scaling the step-size by ϵ. Subsequently s'(h) is found by rearranging the expression and taking the limit as $\epsilon \to \infty$.

3.3 AdaGrad: Treat Massive, Rare and Sparse Data

In section 3.1.2 the effect of the step size on convergence is discussed. For time-independent parameters, $\boldsymbol{\beta}$, equations (3.9) and (3.10) show that fixed step sizes, α, induce a bias $O(\alpha)$ in the expected risk, while SGD with optimally annealed step-sizes can have the same assymptotic efficiency as batch gradient descent. For large data sets a new problem with standard SGD arises; assigning the same step-size to each coefficient of the parameter estimates can be suboptimal for large parameter spaces.

In some contexts rare data points can be more informative than common ones. In text analysis, for instance, several methods[19]. build on the observation that uncommon words in sentences tend to lend more meaning to phrases, than common words. For instance, the phrase 'the university is very big' illustrates that the uncommon word 'university' is arguably more important for the message of the phrase than the common words 'the' and 'is'. The consequence of having the same step size for all parameter estimates is that the method does not weight scarce data more than common data. This can be a problem when training a model on text data for instance, though the problem is general to high dimensional parametric estimation.

Duchi et al. (2011) introduce Adaptive Gradient (AdaGrad) algorithms to address this problem. For time-independent parameters, $\boldsymbol{\beta}$, the methods achieve constant regret per parameter dimension as opposed to SGD. Their comprehensive paper construct a class of algorithms from a modification of two online optimisation methods. The first is Nesterov's primal-dual subgradient method with some extensions. The second is a method of many names: proximal gradient, forward-backward splitting, and composite mirror descent. The method that concerns us is a specialisation of composite mirror descent[20], namely projected gradient descent, an itera-

[19]This includes the tfidf mapping of words in a text corpus to vector space [23]

[20]In order to keep the presentation simple the two algorithms modified by Duchi et al. (2010) will not be treated. The interested reader is directed to their paper.

tive method with updates defined as

$$\boldsymbol{\beta}_{k+1} = \prod_{\mathsf{B}}^{A}(\boldsymbol{\beta}_k - \alpha \boldsymbol{A}^{-1}\boldsymbol{g}_k) \tag{3.19}$$

$$= \underset{\beta \in \mathsf{B}}{\arg\min} \|\boldsymbol{\beta} - (\boldsymbol{\beta}_k - \alpha \boldsymbol{A}^{-1}\boldsymbol{g}_k)\|_A \tag{3.20}$$

where $\prod_{\mathsf{B}}^{A}(\mathbf{b})$ projects point **b** onto B by minimising the Mahalanobis norm $\|\cdot\|_A = \sqrt{\langle \cdot, \mathbf{A}\cdot \rangle}$, where $\boldsymbol{A} \succeq 0$, i.e. the matrix is positive semi-definite, and where g are subgradients[21]. Note that projected gradient descent is a generalisation of SGD [26].

Consider the accumulated regret[22] of this method, $\sum_{\kappa=1}^{k} L(\boldsymbol{\beta}_\kappa)$. Numerous upper accumulated regret bounds are presented by Duchi et al. (2010) for generalisations of this method. The regret of this particular method[23] is

$$\sum_{\kappa=1}^{k} L(\boldsymbol{\beta}_\kappa) \leq \frac{1}{2\eta}\|\boldsymbol{\beta}_1 - \boldsymbol{\beta}\|_{\boldsymbol{A}}^2 + \frac{\eta}{2}\sum_{\kappa=1}^{k}\|\boldsymbol{g}_\kappa\|_{\boldsymbol{A}^{-1}}^2. \tag{3.21}$$

Minimising this upper bound yields the following minimisation problem[24]

$$\min_{\boldsymbol{A}} \sum_{\kappa=1}^{k} \langle \boldsymbol{g}_\kappa, \boldsymbol{A}^{-1}\boldsymbol{g}_\kappa \rangle \text{ s.t. } \boldsymbol{A} \succeq 0, \operatorname{tr}(\boldsymbol{A}) < c < \infty. \tag{3.22}$$

[21]Subgradients generalise the notion of gradients to non-differentiable functions. Since non-differentiable objective functions are not considered in the thesis, this will not be discussed further.

[22]Note that the general methods developed by Duchi et al. (2010) include regularisation terms, L_{reg}, meaning that sparsity inducing L1-regularisation can be carried out, an important feature for large-scale learning methods. This has not been treated here to avoid cluttering the presentation of the material.

[23]Explicitly stated in lecture notes for the *Machine Learning for Big Data* course at the University of Washington [12].

[24]The trace condition ensures that the trace is not indefinitely increased to minimise the objective function.

Solving equation (3.22) yields

$$\boldsymbol{A} = \boldsymbol{G}_k = c\left(\sum_{\kappa=1}^{k} \boldsymbol{g}_\kappa \boldsymbol{g}_\kappa^\top\right)^{1/2} . \tag{3.23}$$

Note that the outer product matrix, $\boldsymbol{G}_k$, depends on the number of examples, k, and is computed iteratively in equation (3.20).

Recall that this method is especially intended to address large parametric spaces where the issues with fixed step sizes are largest. In these cases it is cumbersome to compute $\boldsymbol{G}_k$. To circumvent this problem the AdaGrad method sets

$$A = \operatorname{diag}(\boldsymbol{G}_k), \tag{3.24}$$

which can be computed in linear time, implying that the iterative weight update scheme becomes

$$\boldsymbol{\beta}_{k+1} = \underset{\boldsymbol{\beta}\in\mathsf{B}}{\arg\min} \|\boldsymbol{\beta} - (\boldsymbol{\beta}_k - \alpha \operatorname{diag}(\boldsymbol{G}_k)^{-1}\boldsymbol{g}_k)\|_{\operatorname{diag}(\boldsymbol{G}_k)}, \tag{3.25}$$

and in Euclidean space, for $\mathsf{B} = \mathbb{R}^p$, this reduces to

$$\boldsymbol{\beta}_{k+1,i} = \boldsymbol{\beta}_{k,i} - \eta_{k,i}\boldsymbol{g}_{k,i} \tag{3.26}$$

$$= \boldsymbol{\beta}_{k,i} - \frac{\eta}{\sqrt{\sum_{\kappa=1}^{k} \boldsymbol{g}_{\kappa,i}^2}}\boldsymbol{g}_{k,i} \tag{3.27}$$

this method is implemented in Vowpal Wabbit and enjoys the following upper accumulated regret bound

$$\sum_{\kappa=1}^{k} L(\boldsymbol{\beta}_\kappa) \leq 2R_\infty \sum_{i=1}^{d} \|\boldsymbol{g}_{1:k,j}\|_2 \tag{3.28}$$

where $\boldsymbol{g}_{1:k} = [\boldsymbol{g}_1, ..., \boldsymbol{g}_k]$ is a matrix of the concatenated subgradient sequence, and R_∞ is the largest distance the estimates $\boldsymbol{\beta}_k$ reach from the true parameters $\boldsymbol{\beta}$

$$R_\infty = \max_k \|\boldsymbol{\beta}_k - \boldsymbol{\beta}\|_\infty . \tag{3.29}$$

3.4 Online Normalisation

A disadvantage of standard Stochastic Gradient Descent is that it is sensitive to feature scaling, or the scaling of the observed dependent variables. Unscaled dependent variables can impair the convergence properties of the algorithm [22]. This also applies to the batch Gradient Descent algorithm. A common remedy in the batch setting is to rescale the range of the p-dimensional dependent variables, often to $[0, 1]^p$. These methods are useful, though they become cumbersome for large data sets. In an online learning setting the data arrives in a continuous stream, and it is therefore not evident how one can carry out sequential pre-processing steps that include rescaling.

Ross et. al (2013) modified the standard SGD and AdaGrad algorithms such that the sequential updates were equivalent to updates on feature scaled batch data. The Normalised AdaGrad (NAG) algorithm presented in Algorithm 1 is scale invariant, in the sense that it adapts to arbitrarily scaled data. The analysis of the method is done on similar, but not identical algorithms.

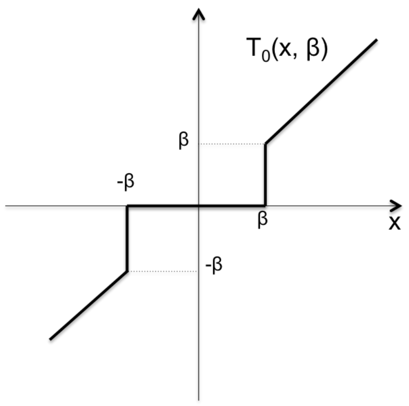
The algorithm has several useful properties. It sequentially scales incoming data while reducing test-time and test-space complexity. Overall this yields a more robust algorithm. Online normalisation is also implemented in Vowpal Wabbit.

Note that though Ross et. al (2013) achieve good results on several real data sets, the methods are not designed in general for data sets from nonstationary generative models with time-dependent parameters, $\boldsymbol{\beta}$. This is because nonstationary data either break the assumptions in the derivations of the upper regret bounds of the methods or could make the bounds arbitrarily large.

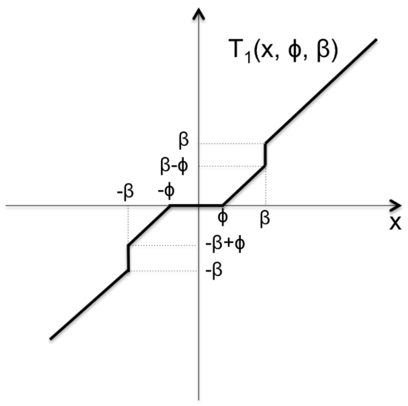
Algorithm 1 NAG - Normalised AdaGrad

```
$\boldsymbol{\beta} \leftarrow \mathbf{0},\ \ \boldsymbol{s} \leftarrow \mathbf{0},\ \ \boldsymbol{G} \leftarrow \mathbf{0},\ \ N \leftarrow 0$          Initialisation
for each k do
    observe (x,y)

    for each i in length(x) do
        if $|x_i| > s_i$ then                          Rescale needed
            $\beta_i \leftarrow \beta_i \frac{s_i}{|x_i|}$                          Rescale
            $s_i \leftarrow |x_i|$                              Update scale
        end if
    end for

    $\widehat{y} \leftarrow \sum_i \beta_i x_i$                              Update estimates
    $N \leftarrow N + \sum_i x_i^2 / s_i^2$                       Update $N$

    for each i in length(x) do
        $G_i \leftarrow G_i + \left(\frac{\partial L(\widehat{y},y)}{\partial \beta_i}\right)^2$               Squared gradient sum
        $\beta_i \leftarrow \beta_i - \eta \sqrt{\frac{k}{N}} \frac{1}{s_i \sqrt{G_i}} \frac{\partial L(\widehat{y},y)}{\partial \beta_i}$       Update estimates
    end for

end for
return $\boldsymbol{\beta}$
```

3.5 Truncated Gradient: Online Sparsification



(a) Truncation function, T_0, used for *simple coefficient rounding*.



(b) Truncation function, T_1, used for *truncated gradient*.

Figure 1: Illustrations of truncation functions T_0 and T_1 defined in the text.

Though the variations of stochastic gradient descent discussed thus far are considerably more computationally feasible for large data sets than traditional batch learning techniques discussed in section 2.2, they are not exempt of computational challenges [17].

One important weakness is that the online learning algorithms discussed so far induce non-zero weights on practically all components of the regression coefficient vector. This introduces at least two problems. Firstly *space constraints* related to storing the coefficients into memory can in the worst case cause the RAM[25] to overflow thereby preventing the algorithm to run efficiently. Secondly, the *speed* of the algorithm slows down as the number of non-zero elements in the coefficient vector increases. Both problems can be addressed by reducing the number of

[25]Random-access memory.

non-zero components in the coefficient vector, $\widehat{\boldsymbol{\beta}}$, i.e. through sparsification.

It is undesirable to sparsify with a black-box wrapper solution that iteratively tests the regret induced by nullifying components in the coefficient vector. For large data sets this can be especially computationally infeasible.

Langford et al. (2009) propose using Truncated Gradient, an online sparsification method akin to L1-regularization for a batch learner. The method builds on the Simple Coefficient Rounding method.

Simple coefficient rounding. A naïve, but informative method for sparsifying the coefficient vector is to simply round the coefficients that are below a threshold, $\theta > 0$, to zero, after every K steps of an online learning algorithm. So, for every k^{th} example one updates the coefficient vector using equation (3.1) unless $k/K \in \mathbb{Z}$, in that case one uses the following update rule

$$\boldsymbol{\beta}_{k+1} = T_0\big[\boldsymbol{\beta}_k - \alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_k, \widehat{\boldsymbol{z}_k} | \boldsymbol{\beta}_k), \theta\big], \tag{3.30}$$

where we define the function T_0 as

$$T_0[\beta_j, \theta] = \begin{cases} 0, & |\beta_j| \leq \theta \\ \beta_j, & \text{otherwise.} \end{cases} \tag{3.31}$$

This truncation function in equation (3.31) is illustrated in Figure 1a. There are two main problems using this truncation function. Firstly, the method is sensitive to the choice of K, since many choices lead to adverse performance effects. The second is the lack of a theoretical guarantee for the method's performance in an online setting. To address these issues, Langford et al. (2009) propose a less aggressive truncation that does not directly round to zero below a threshold of the function argument. The method is termed *truncated gradient.*

Truncated gradient. The proposed method performs standard Stochastic Gradient Descent steps, as defined by equation (3.1), for iterate, k, where $k/K \notin \mathbb{Z}$ and otherwise

$$\boldsymbol{\beta}_{k+1} = T_1\big[\boldsymbol{\beta}_k - \alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_k, \widehat{\boldsymbol{z}_k}|\boldsymbol{\beta}_k), \alpha_k g_k, \theta\big], \tag{3.32}$$

where $g_k > 0$, is the gravity parameter controlling the shrinkage of the truncation function, T_1, which is defined as

$$T_1[\beta_j, \phi, \theta] = \begin{cases} \max(0, \beta_j - \phi), & \beta_j \in [0, \theta], \\ \min(0, \beta_j + \phi), & \beta_j \in [-\theta, 0], \\ \beta_j, & \text{otherwise.} \end{cases} \tag{3.33}$$

Note that Figure 1b illustrates the truncation function and how the shrinkage depends on $\phi = \alpha_k g_k$.

Langford et al. (2009) prove a theorem[26] that bounds the regret of this approach such that the regret increases with sparsity and conversely. Empirical results from an implementation in Vowpal Wabbit back the validity and efficiency of the method.

[26]The theorem can be specialised to specific loss functions. As stated, it does not apply to annealed step-sizes. The authors claim, however, that the result can be extended to annealed step-sizes too.

3.6 Model Assessment for Online Learning

Vowpal Wabbit has implemented progressive validation, a modern method of assessing the performance of its sequential optimisation algorithms. This section seeks to cover foundational concepts for assessing and selecting statistical models, culminating in an exposition of progressive validation.

3.6.1 Foundational Concepts for Batch Learning

Error. The purpose of building a statistical model based on observed data is for its predictions to *generalize* to independent test data [13]. A common approach for achieving this is to minimize what is termed the *generalization error* or the *test error* of an independent test sample[27]. This is evaluated as the expected loss of the loss function on independent test observations, i.e. the risk in equation (2.10). Estimating the expected loss over the training data $\{y_i\}_{i=1}^{N_{\text{train}}}$yields what is termed the *training error*, R_{train}

$$R_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} L(y_i, \widehat{y_i}). \tag{3.34}$$

Note that the training error is not always a good estimate of the test error. Increasing the model complexity can reduce the training error to zero. A model with an extremely low training error can be overfit to the data and generalize poorly.

Error Decomposition. When comparing different models it is often useful to decompose the generalization error, R. For squared error loss

[27] I.e. a sample with observations that are not used to choose the model type or to estimate the parameters of the model.

the decomposition yields

$$\begin{aligned} R &= E\big[(Y - \widehat{Y})^2\big] \\ &= \mathsf{Var}(Y) + \mathsf{Var}(\widehat{Y}) + \mathsf{Bias}(\widehat{Y})^2. \end{aligned} \tag{3.35}$$

and shows that the generalization error consists of three components: the irreducible error Var(Y), the model's bias squared Bias$(\widehat{Y})^2$ and the model's variance Var$(\widehat{Y})$. The decomposition of the generalization error can be made for different loss functions yielding different weights for the components [10].

The error decomposition is informative as it highlights the bias-variance trade off of a model. One may for instance reduce a model's bias by increasing the model's complexity, often at the cost of increasing the model's variance. This would not reduce the test error in general.

Model selection and assessment. The construction of statistical models can be divided into two important phases. The first is model *selection* among a number of different models. The last is model *assessment* of the prediction or generalization error of the selected model. Hastie et. al (2001) state that the best approach for completing these two phases in a data-rich situation is to randomly partition the data into separate *training*, *validation* and *test* sets. The candidate models are fit to the data in the training data. Subsequently the prediction error of the candidate models are estimated on the independent validation set. The best performing candidate model in the validation is selected and its generalization error is estimated on the data in the independent test set.

3.6.2 Cross Validation

Cross validation is a widespread model assessment method for estimating how well a statistical model generalizes to independent test data in a batch setting [13]. Most commonly it is used to assess a model's predictive capabilities. This is done by partitioning the data into separate training and test sets[28].

K-fold cross validation. The data is randomly split into K approximately equal parts. The data in $K-1$ of the sets is used to train the model, and 1 set is used as a test set on which the generalization error is estimated. In order to reduce the variance of the generalization error estimate, this training and testing is repeated K times such that all partitions are used as a test set once.

The generalization error estimated by the cross validation can be expressed as

$$R_{\text{CV}} = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, \widehat{y_i}^{-\kappa(i)}\right) \tag{3.36}$$

where $\kappa : \{1, ..., N\} \to \{1, ..., K\}$ is an indexing function indicating which partition an observation has been allocated to, and where $\widehat{y_i}^{-k}$ is a response fitted on the data with the k^{th} partition removed.

Note that the edge cases when $K = N$ and $K = 2$ are referred to as *leave-one-out* cross validation and the *holdout method*, respectively.

Exhaustive cross validation. The K-fold cross validation method outlined above is a non-exhaustive cross validation method as the generalization error is not based on all possible partitions of the data in $K-1$ training sets and 1 test set. However, since the scope of this thesis is primarily online learning, this model assessment method, which is primarily used to assess learning algorithms in a batch setting, will not be discussed further.

3.6.3 Progressive Cross Validation

Blum et al. (1999) propose a procedure termed *progressive* cross validation that has similar error bounds as the holdout method of section 3.6.2, but scales better in an online learning setting.

[28]The method can be extended to include the validation sets in a data-rich setting.

Procedure. One starts by randomly dividing the data into a training set and a test set. The model is then fit on the training data. For each of the $i = 1, ..., n_{\text{test}}$ data points in the test set, the model is trained on the training data *and* on the $i-1$ previous data points in the test set, before it is tested on the i^{th} data point in the test set to yield[29] an error estimate $\widehat{e_i}$ of the generalization error $\bar{e}_i$. The model output is randomly selected among the n_{test} models considered in the test set, with an error estimate computed by averaging the error of the n_{test} tests, $\widehat{e}_P$.

$$\widehat{e}_P = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \widehat{e_i}. \tag{3.37}$$

Note that $\widehat{e}_P$ is an estimate of the generalization error $\bar{e}_P = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \bar{e}_i$. This error estimate can be used to continuously monitor the performance of an online learner. The error is implemented for the experiments considered in this thesis, thereby shedding light on how well it can accomplish the goal of tracking a learner's performance. Ideally it should be able to detect model degeneration.

Hoeffding bound. Hoeffding (1963) introduced upper bounds for the probability that the sum, S, of n independent random variables, exceed some positive number t, $P\big(S - \mathrm{E}(S) \geq nt\big)$. Bounds on probabilities of this form are now called *Hoeffding bounds*, and are useful in an array of settings. For instance, the progressive validation error estimate of equation (3.37) is a sum of independent random variables.

Blum et al. (1999) show that the progressive validation error estimate satisfy the same Hoeffding bounds as that of the holdout method with an equivalent test set, namely

$$P\big(|\widehat{e}_P - \bar{e}_P| > a\big) \leq e^{-2a^2 n_{\text{test}}}. \tag{3.38}$$

[29]The error estimate is obtained from the chosen loss function.

4 Design of Experiments

The purpose of the following experiments is to examine some of the properties of the online learning algorithms implemented in `vw`. Note that in an online learning paradigm data arrives continuously in a stream. Therefore it is natural to assume that the generative distribution of the data is nonstationary. The goal of the online model fitting will be to examine how `vw` can adapt simple models in the face of a given nonstationary data stream. This deliberate model misspecification in the dynamic setting is motivated by the fact that in the face of a continuous data stream from a live website or mobile application there will always be model misspecification. The question this thesis seeks to answer is to which extent online learners can thrive in a nonstationary environment riddled with noise and unavoidable model misspecification. An overview of the workflow necessary to implement the experiments is provided in Appendix A.

4.1 Models

Some experiments have been devised to compare the strengths and weaknesses of the discussed methods in both a static setting[30] and a dynamic setting[31]. A simple linear model with constant parameters is considered for the static setting. For the dynamic setting we consider simple time series models.

In the static experiments, particular interest will be paid to the performance of batch versus stochastic gradient descent and some initial experiments are devised to give an intuition for the effect of the number of parameters in the model, p. In general the influence of the following parameters are considered: step size, α, the number of data points, n and the signal to noise ratio, R, of the generative models. It would have been

[30]I.e. a generative model with constant parameters, $\boldsymbol{\beta}$.

[31]I.e. a generative model with time-dependent parameters $\boldsymbol{\beta} = \boldsymbol{\beta}(t)$.

interesting to examine the effect of the mini-batch size, m, but this option has been unavailable in vw for some time[32]. It was deemed unnecessary to code experiments from scratch for this scenario. We expect that increasing the mini-batch size to reduce the variance of the training errors, and see the performance of the online learner approach the performance of the batch learner.

4.1.1 Simple Linear Model

The following experiments, based on the following stationary linear models, aim to illuminate the properties of the vw learner. However, it is important to realise that these experiments are unrealistic for at least two reasons. Firstly, one expects real world data to be non-stationary, and secondly one expects there to be considerable misspecification between the estimated model and the generative model of the data. In this section we consider experiments despite these flaws, to build intuition about Vowpal Wabbit before examining experiments with non-stationary data and model misspecification in section 4.1.2.

Consider first a simple linear regression setting with true parameters

$$\beta_j \sim N(\mu_\beta, \sigma_\beta^2) \ \forall \ j \in p, \tag{4.1}$$

where the design matrix, $\boldsymbol{X}$, and noise terms, $\boldsymbol{\epsilon}$ are sampled according to

$$\begin{aligned} X_{ij} &\sim \text{Unif}(x_{\text{min}}, x_{\text{max}}), \ \text{ for finite } x_{\text{min}} < x_{\text{max}} \in \mathbb{R} \\ \epsilon_i &\sim N(0, \sigma_\epsilon^2), \end{aligned} \tag{4.2}$$

where $i \in \{1, ..., n\}$ and $j \in \{1, ..., p\}$ such that the responses are sampled from

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{4.3}$$

[32]https://github.com/JohnLangford/vowpal_wabbit/issues/355

while the linear predictor is

$$\boldsymbol{y} = \boldsymbol{X\beta}. \tag{4.4}$$

The model used to fit the stream of observed responses y_k in `vw` is the same simple linear model described above. There is therefore no model misspecification in this static setting.

4.1.2 Time Series Models

Consider the simple time series model from particle filter theory that consists of a hidden signal process and an observable process

$$y_k = \beta_k + e_k \qquad \text{observable process} \tag{4.5}$$

$$\beta_k = \beta_{k-1} + v_k \qquad \text{signal process} \tag{4.6}$$

for $k \in \{1, 2, ...\}$, where $\beta_1 = 0$ and $e_k \sim N(0, \sigma_e^2)$, $v_k \sim N(0, \sigma_v^2)$.

The experiments are carried out by varying the signal to noise ratio, $R = \sigma_e/\sigma_v$, and varying σ_v.

The model used to fit the stream of observed responses y_k in `vw` is a simple linear model with only a intercept. There is therefore, as discussed, deliberate model misspecification in this dynamic setting.

4.1.3 Implementation of Regression

For both experiments we consider a risk function with squared error loss without a regularization term, see equation (2.7). For the batch algorithm implemented in `R`'s `lm` function this yields the following update rule for each iterate k,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \frac{2\alpha}{n} \sum_{i=1}^{n} (y_i - \widehat{y}_i)\boldsymbol{x_i} \tag{4.7}$$

which is modified to the online algorithm, implemented in `vw`, as follows

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \frac{2\alpha}{n}(y_k - \widehat{y}_k)\boldsymbol{x_k} \text{ for each example } k. \tag{4.8}$$

4.2 Experiments

4.2.1 Static Setting

These experiments are based on the Simple Linear Model of section 4.1.1. As previously mentioned, many real data streams facing a statistician are unlikely to be stationary over large periods of time. However, an examination of the performance of online learning in static settings, where there is no model misspecification, is included as informative background material before carrying out experiments in the dynamic settings.

The experiments seek to examine two aspects of online learning in a static setting. The first is to assess the impact of the signal-to-noise ratio[33], R, on the model fitting performance,

$$R = \sigma_{\text{noise}}/\sigma_{\text{signal}}. \tag{4.9}$$

For the simple linear models, the relationship can be simplified by carefully selecting the parameters of the uniform distribution generating X_{ij} and β_j. In Appendix B we derive that

$$R = \sigma_\epsilon/\sqrt{p}\sigma_\beta, \qquad \text{for } \mu_\beta = 0 \text{ and } (x_{\min}, x_{\max}) = (-1, 2). \tag{4.10}$$

which is easy to interpret. Tuning R effectively increases or reduces the amount of noise added to the linear predictor to yield the observed responses. This is exemplified by Figure 2. Three experiments are carried out to assess the effect of R on the performance of the online learner. These are summarised in Table 1.

The second is to examine effect the number of parameters have on the performance of online learning is examined in the experiments summarised in Table 2. In Appendix B the expected value and variance of $\widehat{y}$ are derived

[33]Strictly speaking R is defined as the noise-to-signal ratio, but for convenience we avoid this terminology.

for multiparameter scenarios to be

$$\mathrm{E}[\widehat{y}_i] = 0, \qquad \text{for } \mu_\beta = 0 \tag{4.11}$$

$$\mathrm{Var}[\widehat{y}_i] = \sigma_\epsilon^2 \left(1 + \frac{1}{R^2}\right), \qquad \text{for } \mu_\beta = 0 \text{ and } v_x + \mu_x^2 = 1 \tag{4.12}$$

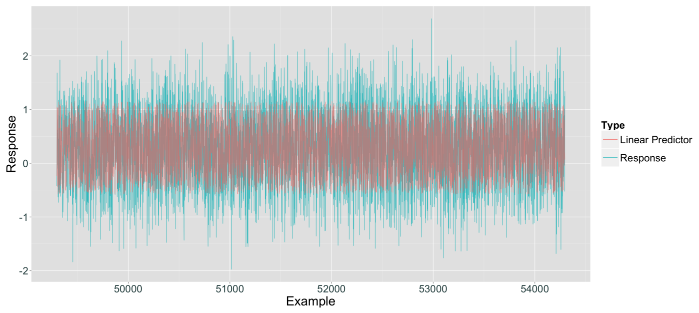
All static experiments are run for $n = 3 \cdot 10^5$ though sample sizes as large as $2 \cdot 10^6$ were tried without changing the conclusions that follow.

Table 1: Parameter settings for the experiments on the signal to noise ratio. R is the signal-to-noise ratio; σ_ϵ is the standard deviation of the noise; μ_β is the mean of the simulated $\boldsymbol{\beta}$ parameters; $x_{\min}$ and $x_{\max}$ are the limits of the uniform distribution from which the independent variables X_{ij} are sampled; n is the sample size and p the number of parameters.

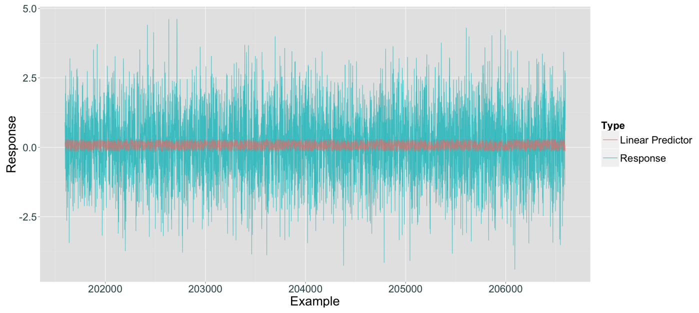
Experiment	R	σ_ϵ	μ_β	$x_{\min}$	$x_{\max}$	n	p
S1	1	1	0	-1	2	$3 \cdot 10^5$	1
S2	10	1	0	-1	2	$3 \cdot 10^5$	1
S3	100	1	0	-1	2	$3 \cdot 10^5$	1

Table 2: Parameter settings for the experiments on the amount of parameters. R is the signal-to-noise ratio; σ_ϵ is the standard deviation of the noise; μ_β is the mean of the simulated $\boldsymbol{\beta}$ parameters; $x_{\min}$ and $x_{\max}$ are the limits of the uniform distribution from which the independent variables X_{ij} are sampled; n are the sample size and p the number of parameters.

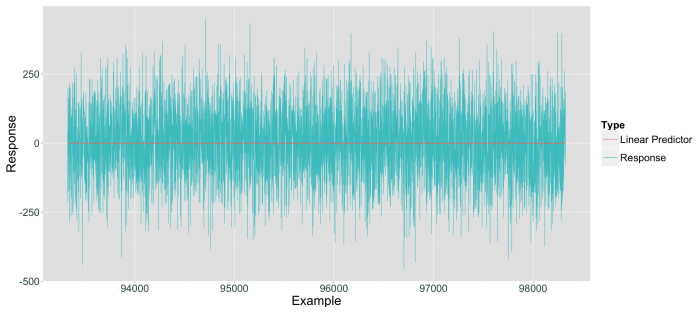
Experiment	R	σ_ϵ	μ_β	$x_{\min}$	$x_{\max}$	n	p
S2	10	1	0	-1	2	$3 \cdot 10^5$	1
S4	10	1	0	-1	2	$3 \cdot 10^5$	20



(a) Experiment S1: Observed responses alongside linear predictor for $R = 1$.

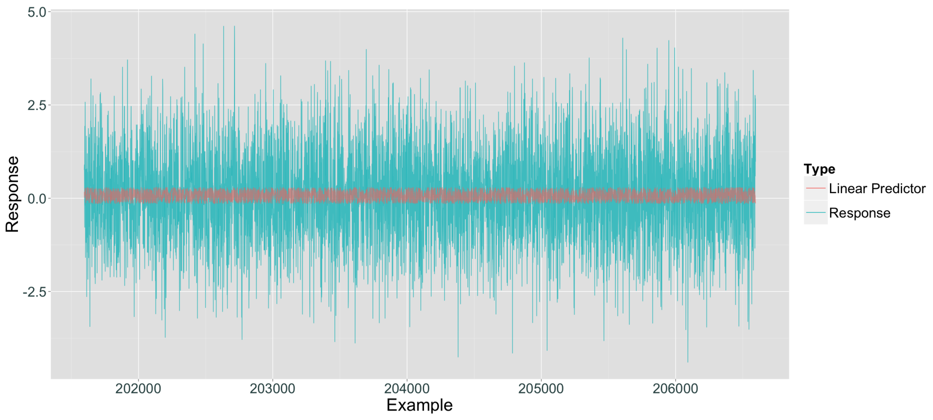


(b) Exp. S2: Observed responses alongside linear predictor for $R = 10$.

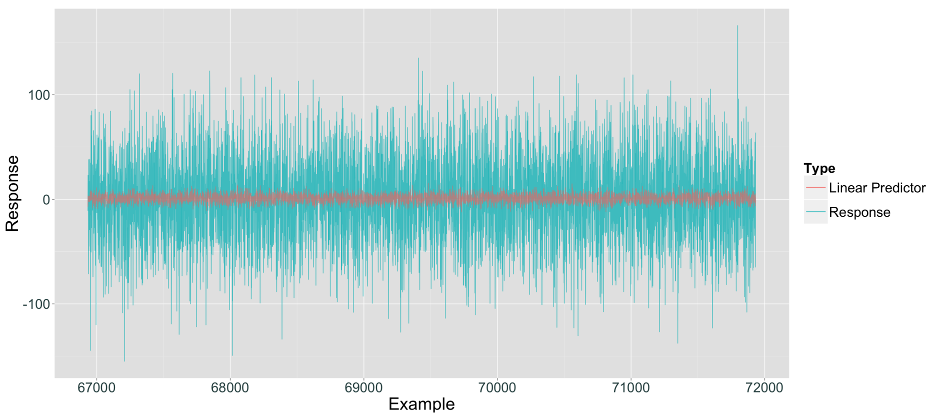


(c) Exp. S3: Observed responses alongside linear predictor for $R = 100$.

Figure 2: The above plots of observed responses and linear predictors demonstrate the effect of increasing the signal to noise ratio R for a stationary generative model. Note that only a random interval of length 10^3 is displayed.



(a) Exp. S2: Observed responses alongside linear predictor for $p = 1$.



(b) Exp. S4: Observed responses alongside linear predictor for $p = 20$.

Figure 3: The above plots of observed responses and linear predictors demonstrate the effect of increasing the number of parameters, p, in the generative model. The absolute value of the observational noise in Experiment S4 is greater than Experiment S2 due to p being larger. However the signal to noise ratio is visibly similar, as expected. Note that only a random interval of length 10^3 is displayed.

4.2.2 Dynamic Setting

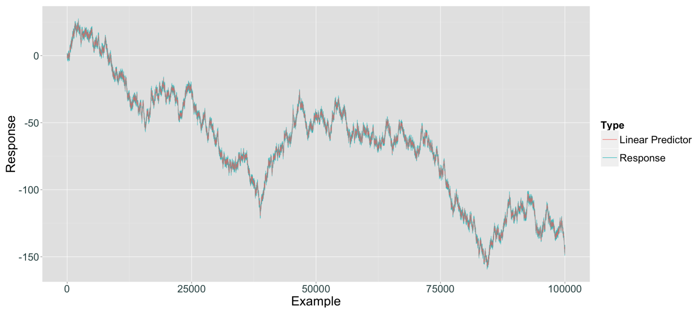
These experiments are based on the Time Series Model of section 4.1.2. The dynamic setting experiments are designed to be more realistic, without of course attempting to fully mimic real data sets. There will be model misspecification between the trained model and the generative model. And the generative models will be nonstationary. It turned out that it was not necessary to have the generative model change over time in order to draw useful conclusions from the experiments. Experiments on data simulated from generative models that change with time are instead included in section 6, which treats an online learning method we have designed to specifically handle nonstationary data from varying generative models.

In the experiments, particular attention is paid to the learning rate, α, the most important factor in determining the success of the model fitting. This is done by carrying out experiments on `vw`'s AdaGrad, and with different fixed learning rates. This is summarised in Tables 3 and 4 respectively. The impact of the signal to noise ratio of the time series model, R, on the performance of `vw` is also considered. From equation (4.6) it follows that

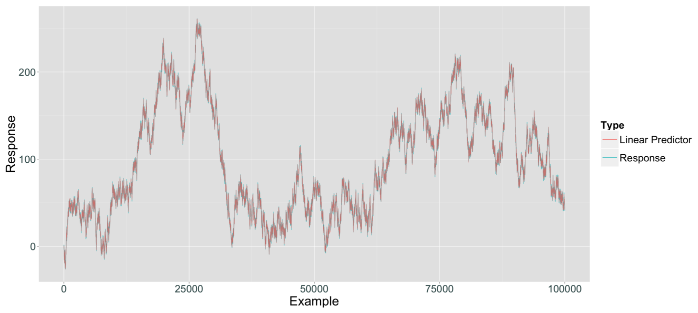
$$R = \sigma_e/\sigma_v. \tag{4.13}$$

Table 3: Parameter settings for the three AdaGrad experiments with varying signal to noise ratio, R. The generative model is the time series of eq. (4.6). The standard deviation of the signal process is σ_v and n are the number of samples.

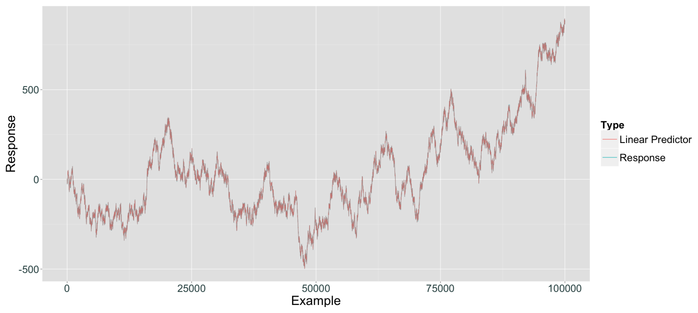
Experiment	R	σ_v	n
D1	0.1	1	10^5
D2	1	1	10^5
D3	10	1	10^5



(a) Experiment D1: Observed responses and linear predictors for $R = 0.1, \sigma_v = 1$.



(b) Exp. D2: Observed responses and linear predictors for $R = 1, \sigma_v = 1$.



(c) Exp. D3: Observed responses and linear predictors for $R = 10, \sigma_v = 1$.

Figure 4: The above plots of observed responses and linear predictors demonstrate the effect of increasing the signal to noise ratio R for a dynamic generative model.

Table 4: Parameter settings for the three experiments with fixed learning rates, α, and a given signal to noise ratio, R, where the generative model is the time series of equation(4.6). Note that σ_v is the standard deviation of the signal process and n are the number of samples. $\alpha = 0.5$ is the default learning rate in `vw`.

Experiment	α	R	σ_v	n
D4	0.005	0.01	1	10^5
D5	0.05	0.01	1	10^5
D6	0.5	0.01	1	10^5

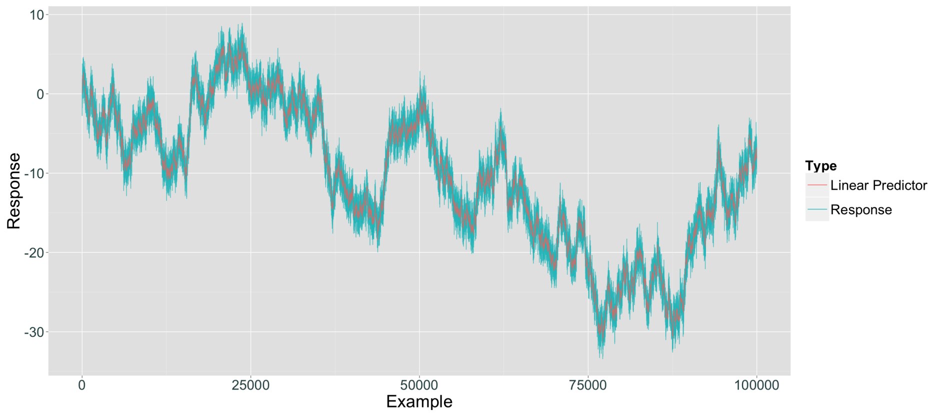


Figure 5: Experiments D4-D6: Observed responses and linear predictors for $R = 0.01, \sigma_v = 1$. A model is fit to the data using three different fixed learning rates, $\alpha \in \{0.005, 0.05, 0.5\}$.

4.3 Model Assessment

The performance of the online learning experiments detailed above will be assessed using a number of key concepts. To avoid misunderstandings, these will be discussed or defined here.

Partitioning of data set. The incoming observations are not placed in either a training or test set. They are dynamically used in both following the Progressive Validation procedure described in section 3.6.3. That is, incoming observations are first used to test the existing model and subsequently to train the model.

Implementation of Progressive Validation error. Due to limited control of `vw` from the terminal, the size of the test set, n_{test}, used to compute `vw`'s Progressive Validation errors, or PV errors, in these experiments is one. For this case there is no need to sample randomly among the first $i-1$ estimates of the test set when facing the i^{th} observation in the test set. This random sampling of estimates would have been impossible to carry out from the terminal interface of `vw`. Instead the PV errors with $n_{\text{test}} = 1$ are computed sequentially for all n examples. This is equivalent to sequentially computing the errors of one-step ahead predictions, a proxy of the test error or the generalization error. One can argue that setting $n_{\text{test}} = 1$ is suboptimal, or that the first observations should not be included in the computation of the PV error. In practise the learners are initialised close to the intial observations, i.e. zero and the results seem meaningful.

Training error. In the following online experiments, the pointwise training error, $\epsilon_{\text{train},i}$, is sequentially computed after the incoming observation, y_i, has been used to train the model, i.e. update the estimate of $\boldsymbol{\beta}$, and thereby estimate the linear predictor, $\widehat{y}_i$ for examples $1 < i < n$.

$$\epsilon_{\text{train},i} = L(\widehat{y}_i, y_i) \tag{4.14}$$

where L is the chosen loss function, in our case, the squared error loss of equation (2.7). Note in some cases L is chosen to be a simple difference operator for the purpose of improving result visualisations. This should be obvious from the plots, as squared error loss yields only positive values, while the difference operator does not.

Estimation error. The advantage of assessing the performance of a learning method on simulated data is that one has access to the generative model, or in our case, the linear predictors, $y_i - \epsilon_i$ for examples $1 < i < n$. This information can be used to sequentially compute the estimation error, $\epsilon_{\text{est},i}$, a more accurate estimate of the model's generalization error.

$$\epsilon_{\text{est},i} = L(\widehat{y}_i, y_i - \epsilon_i) \tag{4.15}$$

Weight error. Another advantage of simulating the data from known model parameters, $\boldsymbol{\beta}$ is that one can sequentially compute the model weight errors, $\boldsymbol{\epsilon}_{\text{weight},i}$, using the parameter estimates, $\widehat{\boldsymbol{\beta}}_i$ for examples $1 < i < n$

$$\boldsymbol{\epsilon}_{\text{weight},i} = \widehat{\boldsymbol{\beta}}_i - \boldsymbol{\beta}_i. \tag{4.16}$$

Note that for the linear models in equation (4.4) we have that $\boldsymbol{\beta}_i = \boldsymbol{\beta}$. Further note that a loss function L was not used to compute the weight error in equation (4.16) for no other reason than to improve visualisations.

5 Results

5.1 Static Setting

The results presented in this section show that `vw` can perform very well in a static setting. This should not be surprising as all the error bounds mentioned in this thesis are formulated for this setting. The experiments in this section are described in section 4.2.1. For details on the experiments on the Effect of Signal to Noise Ratio and the Effect of More Parameters see Table 1 and Table 2, respectively.

5.1.1 Effect of Signal to Noise Ratio

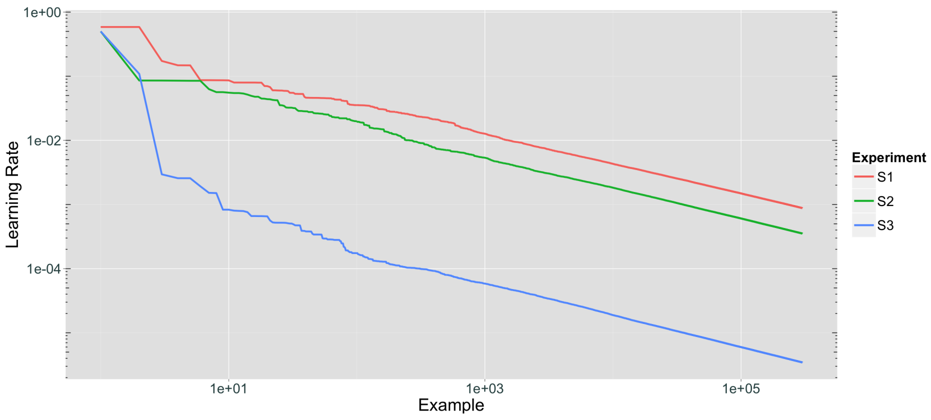
In a static setting `vw` manages to extract the signal from exceedingly noisy environments as can be seen from Figures 7 and 11. AdaGrad has been used for the experiments, implying that the step sizes adapt to the incoming data points as explained in section 3.3. In practice, for this data the step sizes reduced in a relatively stable fashion as demonstrated by Figure 6. Note that for experiments with higher values of R, the learning rates shrink faster.

Varying the signal to noise ratio, R, affects the variance of the training error. Since the estimation errors are relatively small in this static scenario without model misspecification, the training errors are clearly dominated by the noise, as demonstrated by Figures 7 and 10. If one keeps the number of observations, n, unchanged, then increasing R will cause the training errors to be larger overall. Note that although increasing n would cause estimation errors to decrease, this effect would be limited by the learning rates decreasing faster for higher R. Put differently, for higher R less data points are required for the model to converge within the convergence radius due to the convergence radius being larger for high R and the learning rates decreasing faster. The same can be said for the weight estimation errors of Figure 11.

Although the training errors of the two approaches were indistinguishable from each other, the estimation errors and weight estimation errors of `vw` are clearly larger than those of `lm` in Figures 7 and 11. In this sense the speed and scalability of `vw`'s approximative online learning method comes at an accuracy cost.

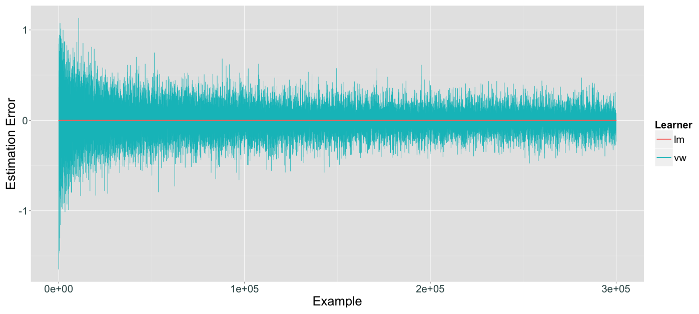
The progressive validation errors explained in section 4.3 are computed and plotted in Figure 8 using a squared error loss. These errors see a slowing decrease, also suggesting model convergence, as expected.

Note that the plots of estimation errors and weight estimation errors per example in Figures 7c and 11c for experiment S3 where $R = 100$ indicate that increasing the signal to noise ratio results in increased estimation volatility. In this sense, and in the sense that the convergence radius of the estimation error is larger for high R, one can say that online learners can struggle in static environments with high signal to noise ratios. Despite this, the overall conclusion is that online learning provides a fast, scalable and accurate method for model fitting in many static scenarios.

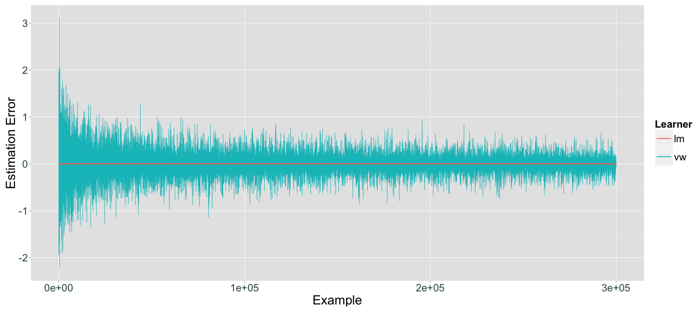


(a) Stable decrease of adaptive learning rates

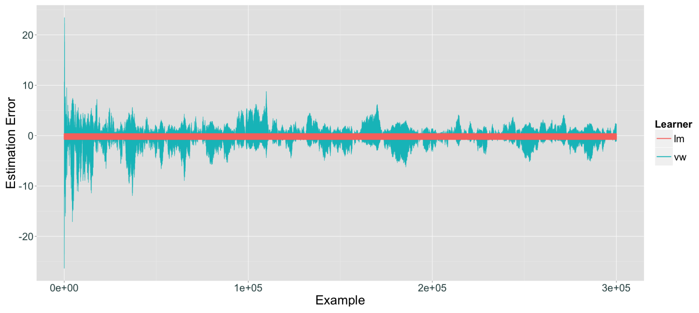
Figure 6: The adaptive learning rates, following the AdaGrad implementation, see a stable decrease per example in this static model setting. Note that the learning rates are decreased faster as the signal to noise ratio, R, increases. For exp. S1-S3 R is $1, 10$ and 100, respectively.



(a) Exp. S1: Estimation errors from online (`vw`) and batch learning (`lm`) approaches. $R = 1$.

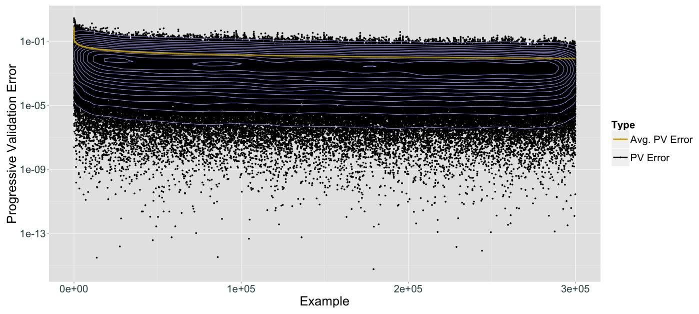


(b) Exp. S2: Estimation errors from online (`vw`) and batch learning (`lm`) approaches. $R = 10$.

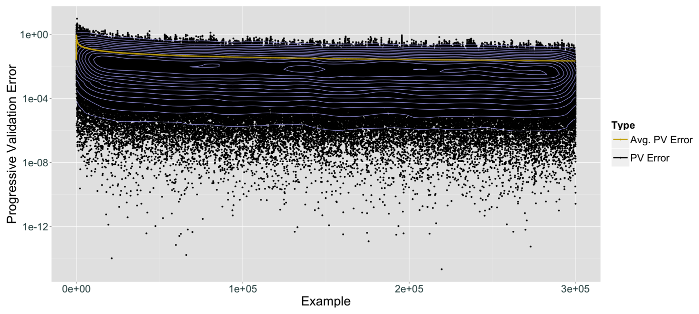


(c) Exp. S3: Estimation errors from online (`vw`) and batch learning (`lm`) approaches. $R = 100$.

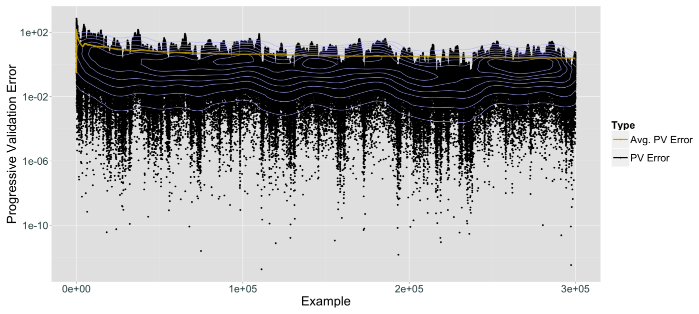
Figure 7: Online and batch estimation errors obtained from `vw` and `R`'s `lm` function, respectively. The online estimation errors show a slowing decrease indicating model convergence for the tested values of signal to noise ratio, R. Higher R increases the volatility of the estimates. Note that the batch estimation errors are minuscule compared to the online estimation errors, as expected for this simple problem with no model misspecification.



(a) Exp. S1: Pointwise and average progressive validation errors. $R = 1$.

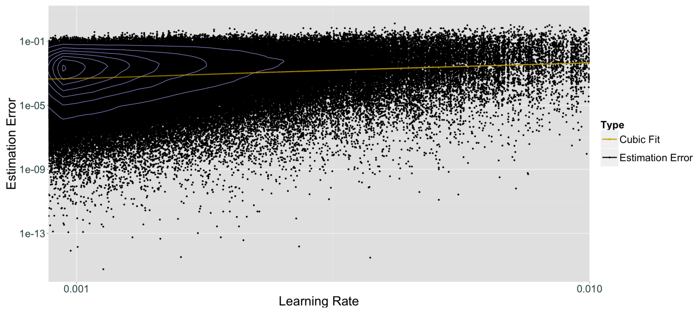


(b) Exp. S2: Pointwise and average progressive validation errors. $R = 10$.

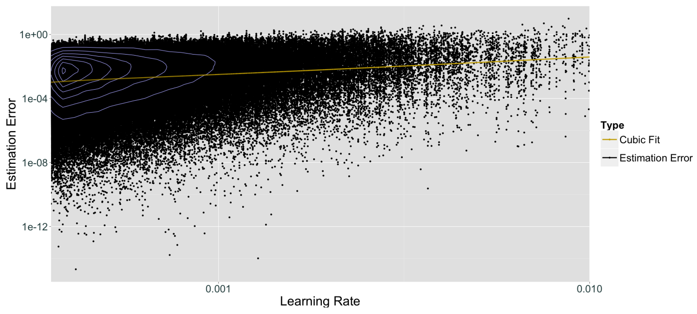


(c) Exp. S3: Pointwise and average progressive validation errors. $R = 100$.

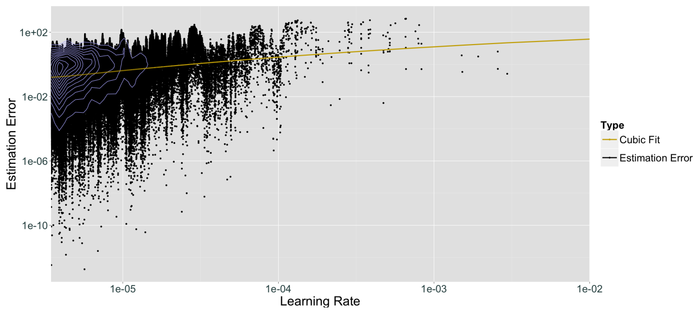
Figure 8: Pointwise and average progressive validation errors for the tested values of signal to noise ratio, R. The stabilising trends suggest model convergence. The blue contour lines indicate the density of the pointwise progressive validation errors.



(a) Exp. S1: Estimation errors plotted against learning rates.

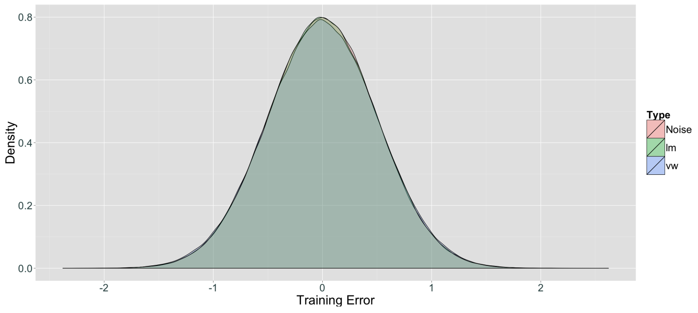


(b) Exp. S2: Estimation errors plotted against learning rates.

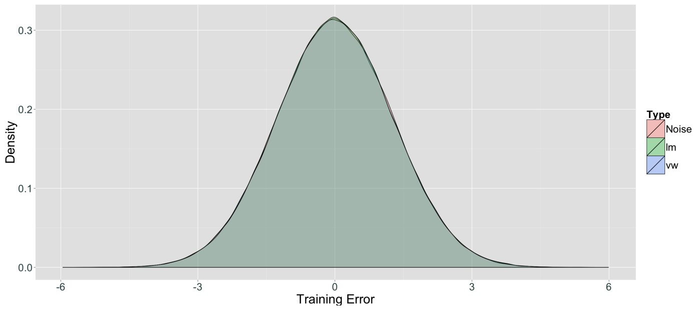


(c) Exp. S3: Estimation errors plotted against learning rates.

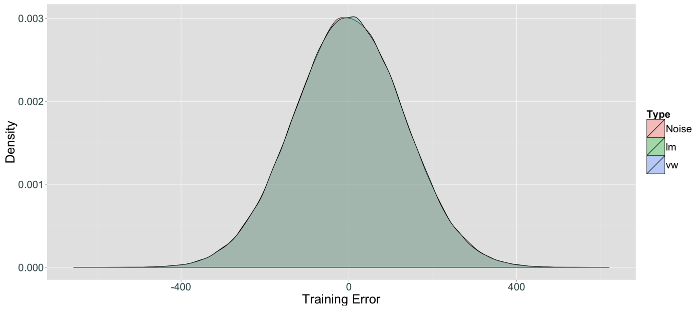
Figure 9: Plotting estimation errors against learning rates indicates that as the learning rates decrease the mean of the errors decreases and stabilises. This indicates model convergence. Experiment S3 indicates that a high signal to noise ratio, R, requires more data for effective model fitting. The effect of increasing the number of data points, n, is however limited by the fast decrease of the learning rates. The blue contour lines indicate the point density of the pointwise progressive validation errors.



(a) Exp. S1: Training errors from online (`vw`) and batch learning (`lm`) approaches. $R = 1$.

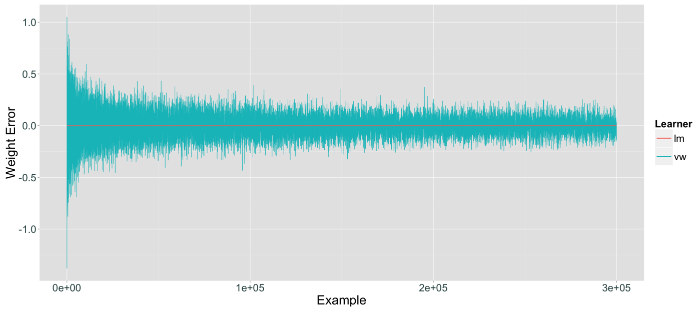


(b) Exp. S2: Training errors from online (`vw`) and batch learning (`lm`) approaches. $R = 10$.

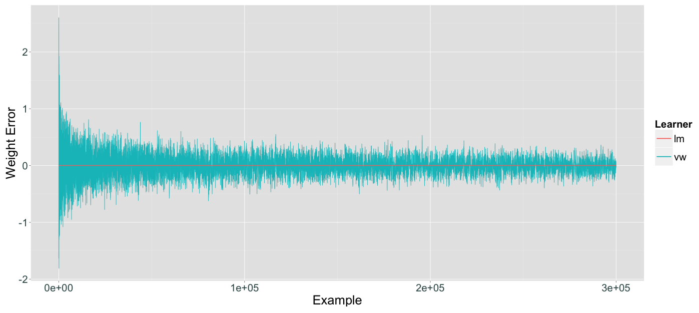


(c) Exp. S3: Training errors from online (`vw`) and batch learning (`lm`) approaches. $R = 100$.

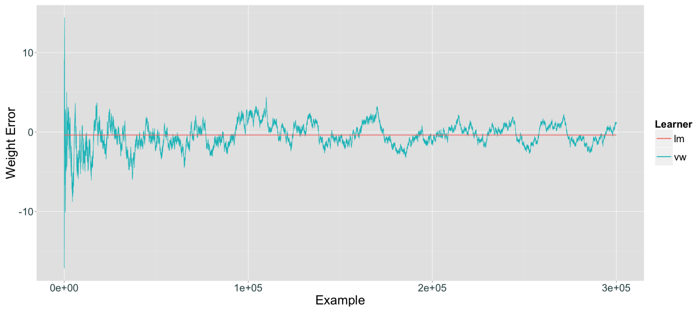
Figure 10: Overlapping density plots of online and batch training errors obtained from `vw` and `R`'s `lm` function, respectively. For the tested values of the signal to noise ratio, R, the online and batch training errors almost completely overlap and follow the probability density function of the normally distributed noise remarkably well. This indicates that most of the training error is caused by observing the variance of the signal's noise, σ_ϵ^2, due to good model convergence. Note that increasing R increases the variance of the training errors, as R is proportional to σ_ϵ.



(a) Exp. S1: Weight estimation errors from online (`vw`) and batch learning (`lm`) approaches. $R = 1$.



(b) Exp. S2: Weight estimation errors from online (`vw`) and batch learning (`lm`) approaches. $R = 10$.



(c) Exp. S3: Weight estimation errors from online (`vw`) and batch learning (`lm`) approaches $R = 100$.

Figure 11: Online and batch weight estimation errors obtained from `vw` and `R`'s `lm` function, respectively. The online weight estimation errors show a slowing decrease indicating model convergence for the tested values of signal to noise ratio, R. Higher R increases the volatility of the weight estimates. Note that the batch weight estimation errors are minuscule compared to the online weight estimation errors, as expected for this simple problem with no model misspecification.

5.1.2 Effect of More Parameters

It is of interest to know how the performance of vw is affected by the number of parameters, p, in the generative and predictive models. AdaGrad has been used for these experiments which, like the previous experiments do not suffer from model misspecification. The step sizes adapt per parameter to the incoming data points in a stable fashion akin to the step size reduction in the one parametric case. This can be seen in Figure 12.

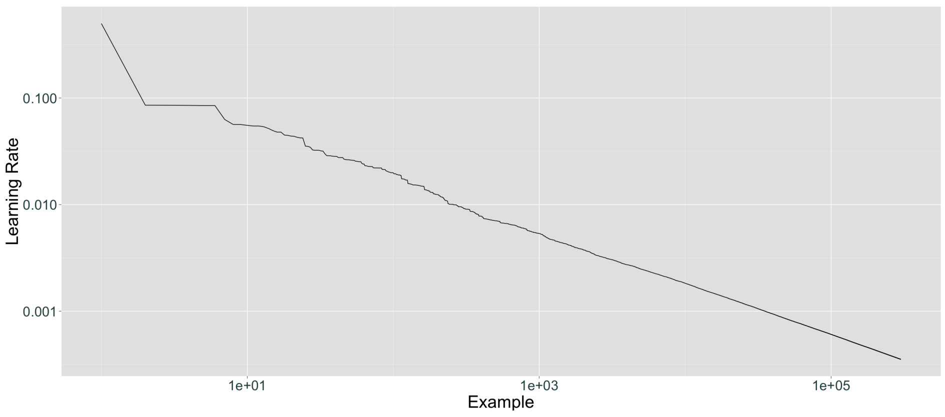
There is in fact little perceived effect from increasing p in the model. Also in the multiparametric case the training errors match the noise of the signal remarkably well, as demonstated by the plots in Figure 13. This implies that one quickly ends up observing the noise of the signal. The same Figure also demonstrates that there is no perceived difference in the training errors obtained from using lm and vw.

The progressive validation errors explained in section 4.3 are computed and plotted in Figure 15 using a squared error loss. These errors see a slowing decrease, suggesting model convergence, as expected. The same can be said of the estimation errors and weight estimation errors in Figures 14 and 17. The estimation error plots in Figure 14 also compare the performance of vw with lm. Although the training errors of the two approaches were indistinguishable from each other, the estimation errors of vw are larger than those of lm. In this sense the speed and scalability of vw's approximative online learning method comes at an accuracy cost.

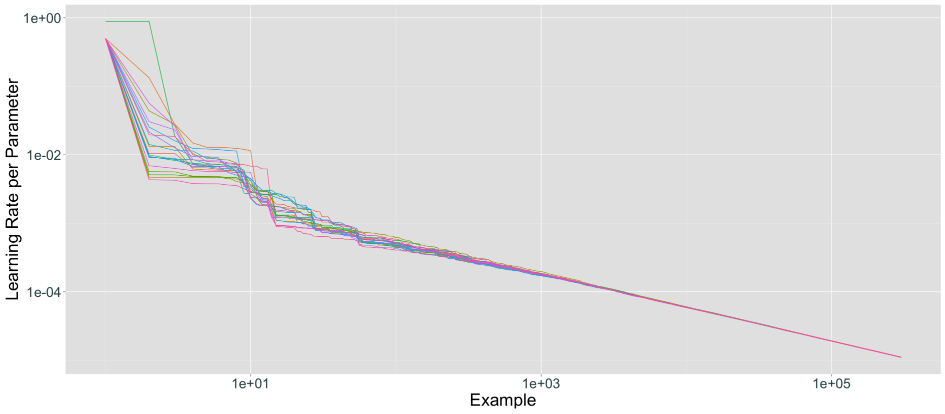
Note that the moderate increase in p has not introduced considerable volatility or instability in the estimates. Overall the performance of vw is very good, though considering that stationary problems tend to be simple problems, this should not be impressive. The focus of this thesis has been primarily on the impact of the learning rate, α, on online learning performance. Otherwise, one could have extended the investigations of this section to much higher values of p, in settings of varying degrees of model misspecification, to assess to which extent SGD suffers from the curse of dimensionality[34].

Another natural extension of these experiments is to examine the performance of `vw` against `lm` when there is model misspecification between the generative and predictive models. However, since this thesis is primarily focused on realistic applications of online learning, this will only be examined in a dynamic setting.

[34]The term curse of dimensionality was coined by RE Bellman (1957) and refers to problems related to the effects of increasing the dimensionality of the parameter space. Big increases to the parameter space cause available data to be sparse, potentially implying that exponential increases to the available data is necessary for reliable results.

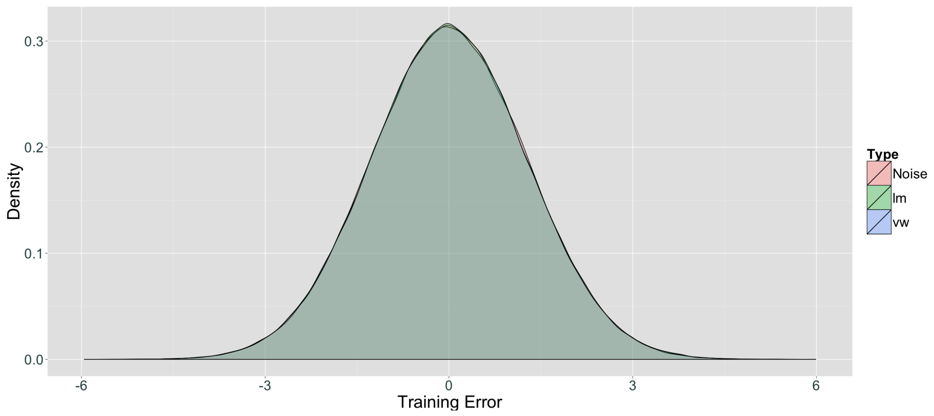


(a) Exp. S2: Stable decrease of adaptive learning rates.

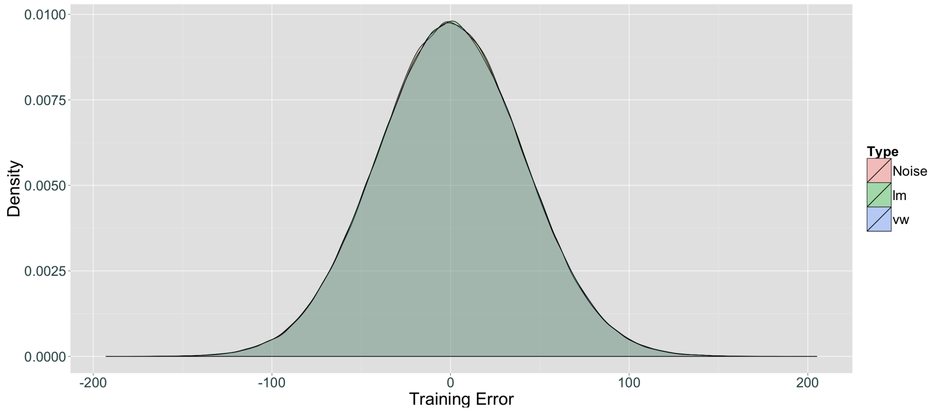


(b) Exp. S4: Stable decrease of adaptive learning rates. Each color represents the learning rates of one parameter.

Figure 12: The adaptive learning rates, following the AdaGrad implementation, see a stable decrease per example in this static model setting. Note that increasing the number of parameters, p, for both the generative and predictive model has an increases the speed of the learning rate reductions. The learning rate reduction speed is comparable across the parameters, as one would expect when there are small differences between the sampling distributions in the p-dimensions.

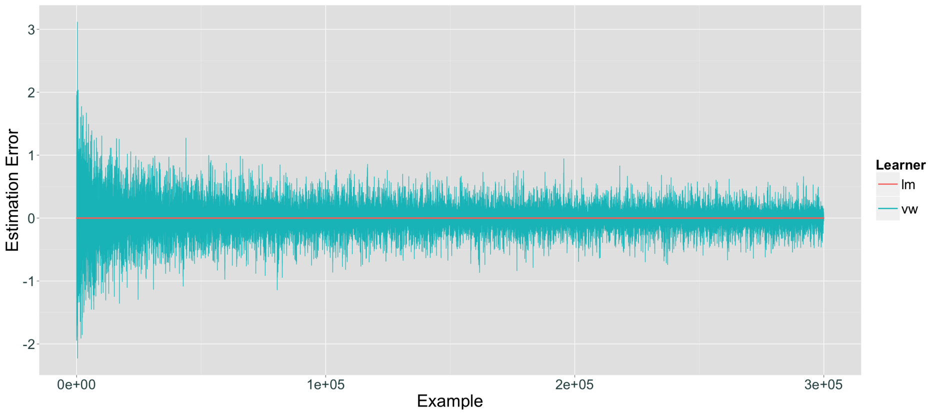


(a) Exp. S2: Training errors from online (`vw`) and batch learning (`lm`) approaches. $p = 1$.

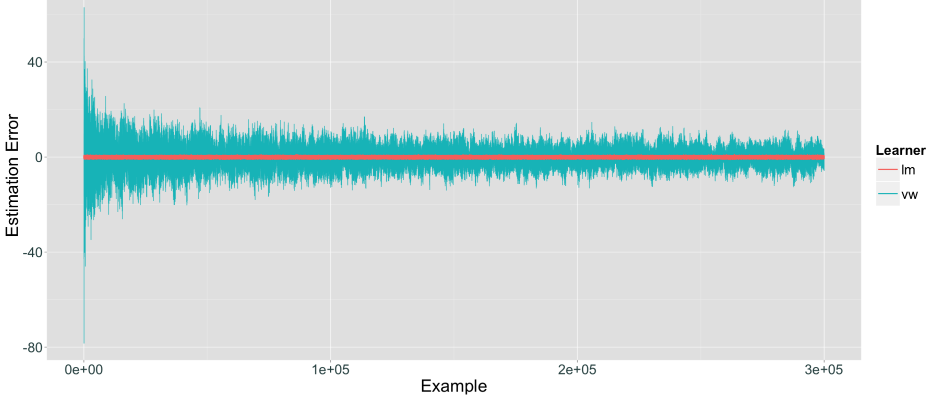


(b) Exp. S4: Training errors from online (`vw`) and batch learning (`lm`) approaches. $p = 20$.

Figure 13: Overlapping density plots of online and batch training errors obtained from `vw` and `R`'s `lm` function, respectively. For the tested values of the number of parameters, p, the online and batch training errors almost completely overlap and follow the probability density function of the normally distributed noise remarkably well. This indicates that most of the training error is caused by observing the variance of the signal's noise due to good model convergence. Note that increasing p for a constant R and signal variance, σ_β^2, increases the variance of the training errors. This follows from the relationship of the variables in equation (4.10). The signal to noise ratio is $R = 10$ for both experiments.

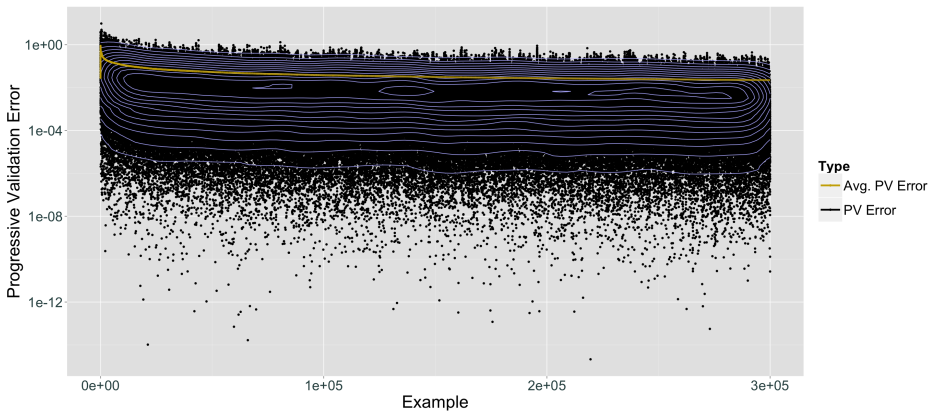


(a) Exp. S2: Estimation errors from online (`vw`) and batch learning (`lm`) approaches. $p = 1$.

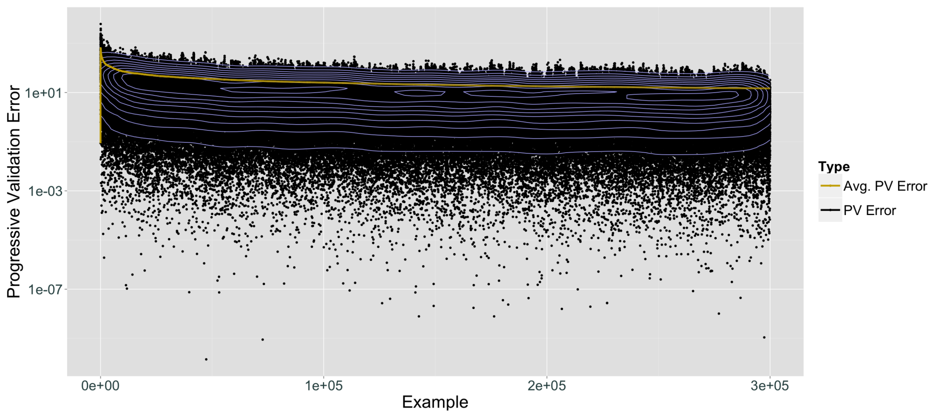


(b) Exp. S4: Estimation errors from online (`vw`) and batch learning (`lm`) approaches. $p = 20$.

Figure 14: Online and batch estimation errors obtained from `vw` and `R`'s `lm` function, respectively. The online estimation errors show a slowing decrease indicating model convergence for the tested values of the number of parameters, p. Note that the batch estimation errors are minuscule compared to the online estimation errors, as expected for this simple problem with no model misspecification.

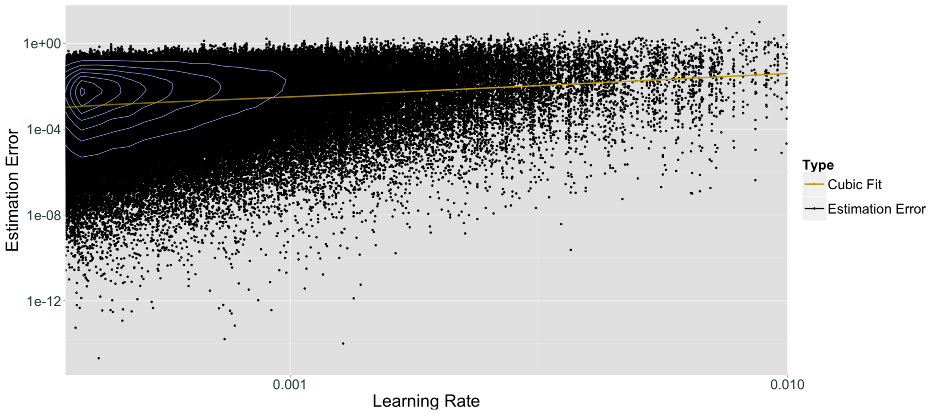


(a) Exp. S2: Pointwise and average progressive validation errors. $p = 1$.

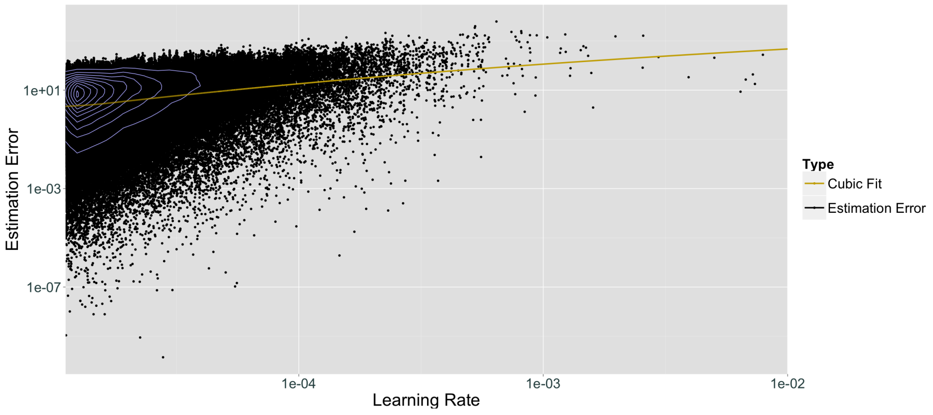


(b) Exp. S4: Pointwise and average progressive validation errors. $p = 20$.

Figure 15: Pointwise and average progressive validation errors for the tested values of the number of parameters, p. The stabilising trends suggest model convergence. The blue contour lines indicate the density of the pointwise progressive validation errors.

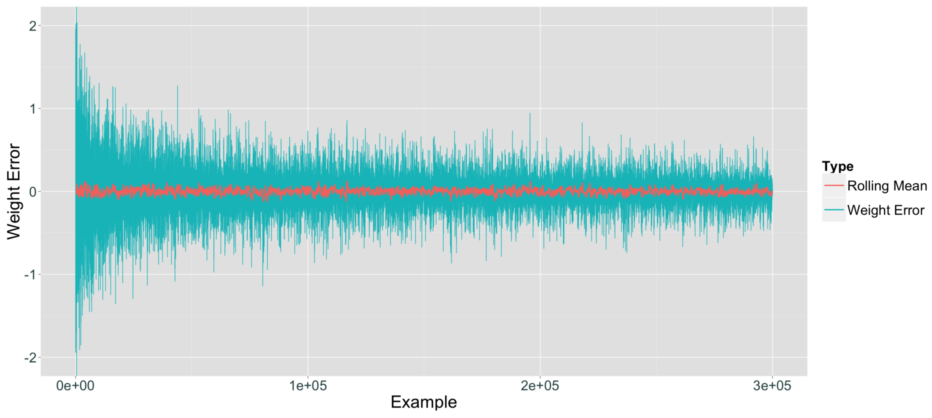


(a) Exp. S2: Estimation errors plotted against learning rates.

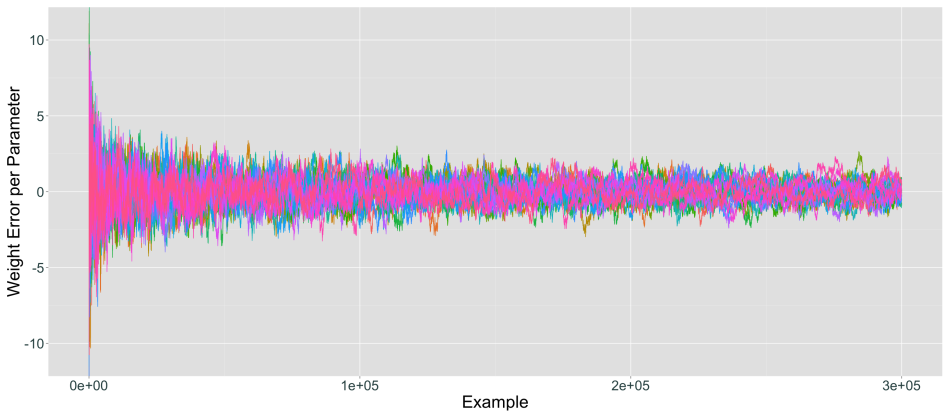


(b) Exp. S4: Estimation errors plotted against learning rates.

Figure 16: Plotting estimation errors against learning rates indicates that as the learning rates decrease the mean of the errors decreases and stabilises. This indicates model convergence. Experiment S4 indicates that a high number of parameters, p, increases the minimum error threshold. This implies that it is more important with larger number of data points, n, in these scenarios, though after a certain point, only minuscule model improvements can be made due to the decaying learning rates. The blue contour lines indicate the point density of the pointwise progressive validation errors.



(a) Exp. S2: Weight estimation errors.



(b) Exp. S4: Weight estimation errors per parameter.

Figure 17: The stability and slowing decrease of the weight errors indicate model convergence. Note that the sizes of the per parameter weight errors are similar for number of parameters, $p = 1$ and $p = 20$. Highly volatile model weight estimates is not a problem for the signal to noise ratio, $R = 10$, in these experiments.

5.2 Dynamic Setting

The time series models considered for these experiments are described in section 4.2.2. The performance of using AdaGrad and Fixed Learning Rates for `vw` on nonstationary input data is assessed in the following. For details about the experiments on the AdaGrad and Fixed Learning Rates see Table 3 and Table 4, respectively.

The results presented in this section show that the performance of `vw` can seriously deteriorate over time in an online setting with nonstationary data. This should not be surprising as the two primary ways of using `vw` on nonstationary data are in fact not specifically designed to handle nonstationary data. Firstly one may naively use AdaGrad, thereby continuously reducing the learning rates and causing the adaptiveness of the model weight estimates to gradually decrease. The second is to set a fixed learning rate, which is suboptimal in an online setting as the parameters of the generative models can change and because `vw` provides no simple way to adaptively choose sensible learning rates for nonstationary data on the fly. This calls for the development of a more flexible method for performing online learning on nonstationary data, which will be discussed in section 6.

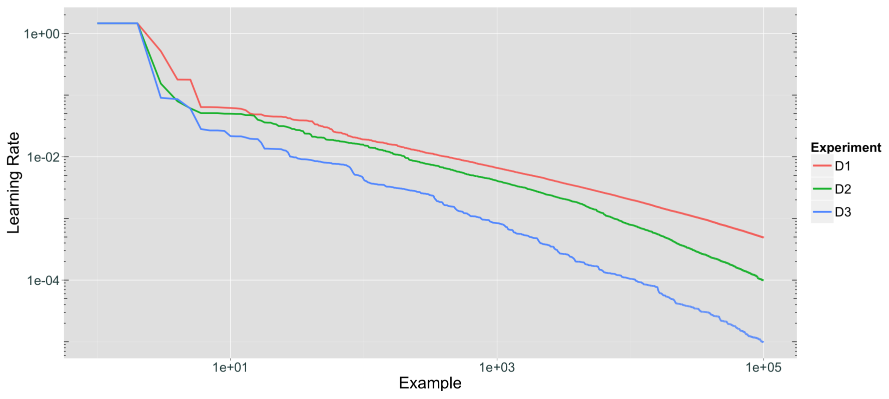
5.2.1 AdaGrad on Time Series Models

The results of these experiments are communicated primarily visually in the plots from Figures 18 to 29.

The key property of the AdaGrad method is that it decreases the learning rates depending on the nature of the incoming data, as explained in section 3.3. This is visualised in Figure 18 for our experiments. Note that for experiments with higher values of the signal to noise ratio, R, the learning rates shrink faster. As expected, reducing the learning rates correlates with increasing the estimation errors in Figure 24.

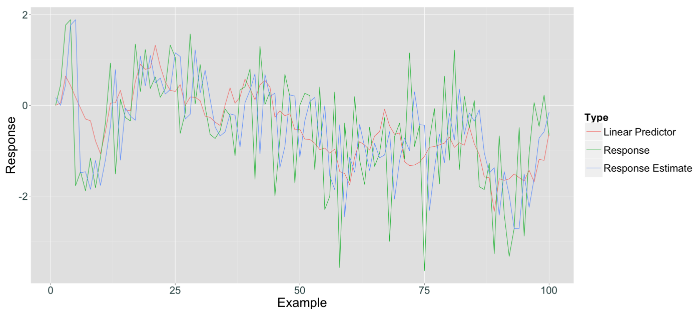
Increasing the signal to noise ratio, R, causes the training errors, estimation errors and weight estimation errors to increasingly deteriorate in Figures 22, 25 and 29, to the point that fitting the model is counterproductive. This is clearly visualised by plots of the first, middle and last 100 response estimates and weight estimates in Figures 19-21 and 26-28 that show how slowly the estimates end up adapting.

Training errors and progressive validation errors in Figures 25 and 23 respectively, are errors the statistician has access to without knowing the generative model of the data or the signal process. The fact that they both show increasing error trends implies that one can detect model deterioration and attempt to approach the problem from a different angle. If one insists on using the online learning methods implemented in `vw` the next natural step would be to test the performance obtained from setting the learning rate to a fixed value. The motivation behind this is that one would then maintain the adaptiveness of the model estimates and prevent model degeneration caused by continuously reducing the learning rates.

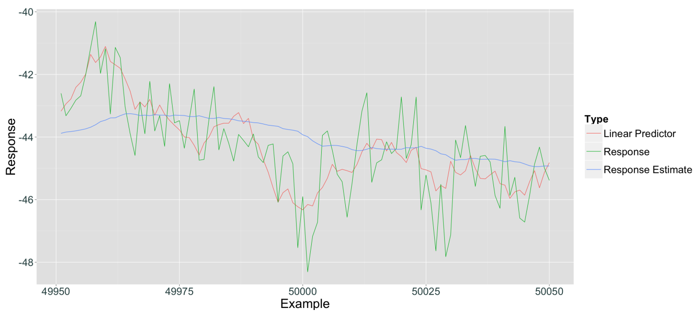


(a) Stable decrease of adaptive learning rates

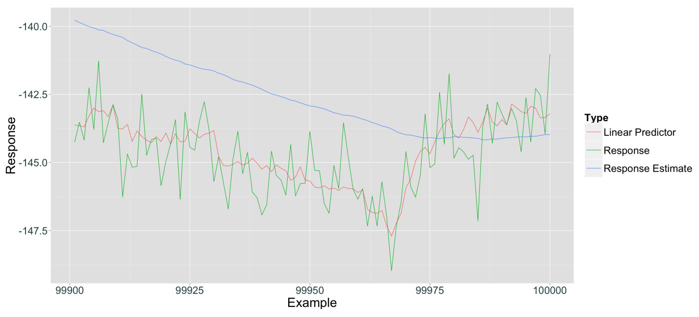
Figure 18: The adaptive learning rates, following the AdaGrad implementation, see a relatively stable decrease per example in this dynamic model setting. Note that the learning rates are decreased faster as the signal to noise ratio, R, increases. For exp. D1-D3 R is $0.1, 1$ and 10, respectively.



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

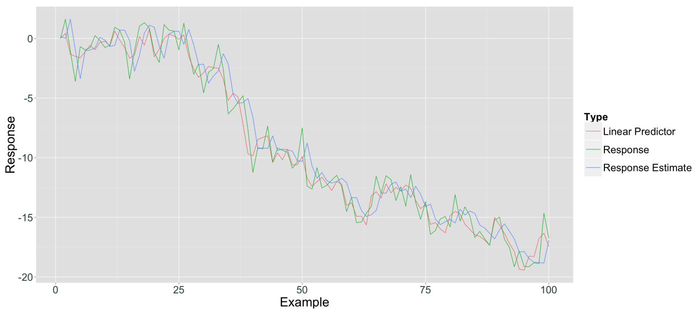


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

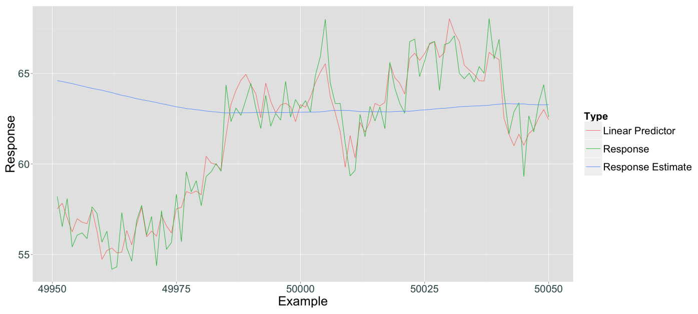


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

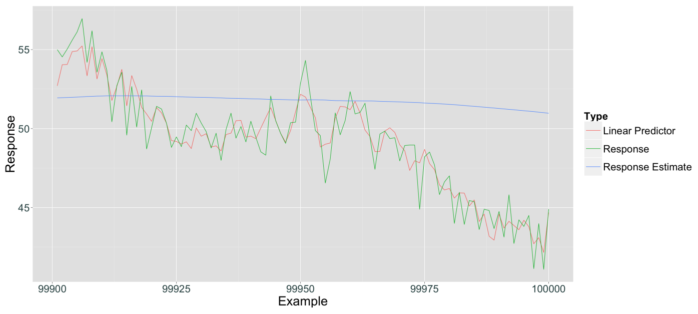
Figure 19: Exp. D1: Plots of response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. It is clear that the adaptability of the estimates strongly decreases over the course of the examples. The estimates slowly become increasingly inaccurate. The signal to noise ratio is $R = 0.1$



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

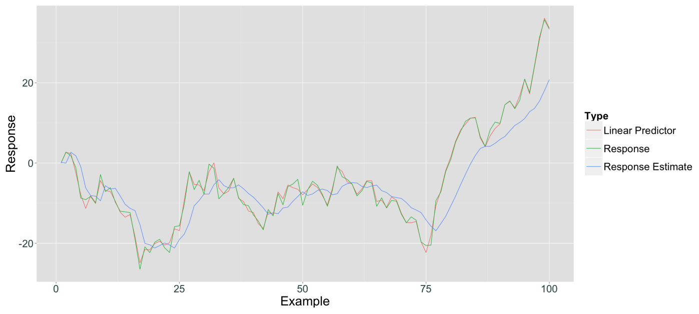


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

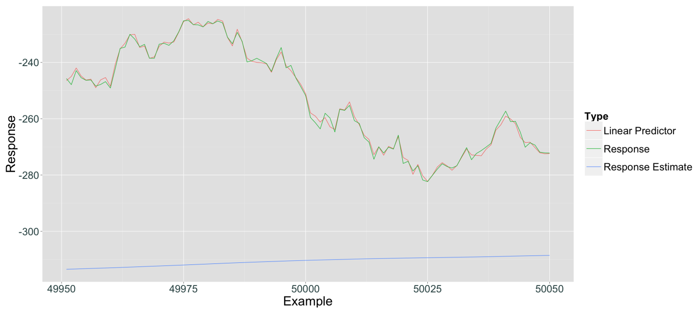


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

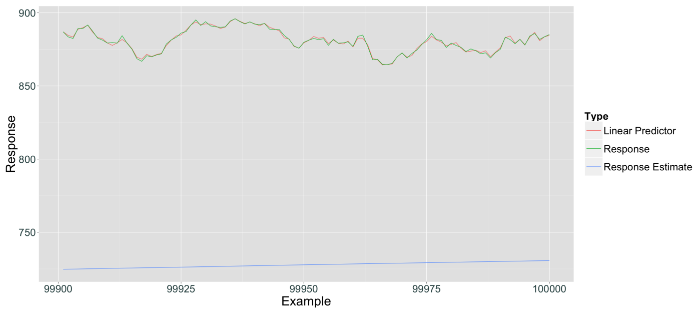
Figure 20: Exp. D2: Response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. It is clear that the adaptability of the estimates strongly decreases over the course of the examples. The accuracy of the estimates deteriorates faster than for Exp. D1 in Figure 19. The signal to noise ratio is $R = 1$.



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

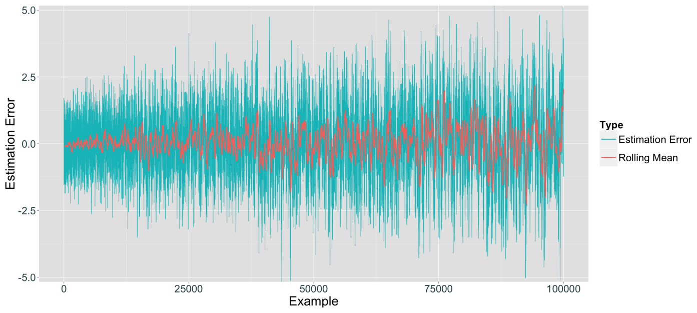


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

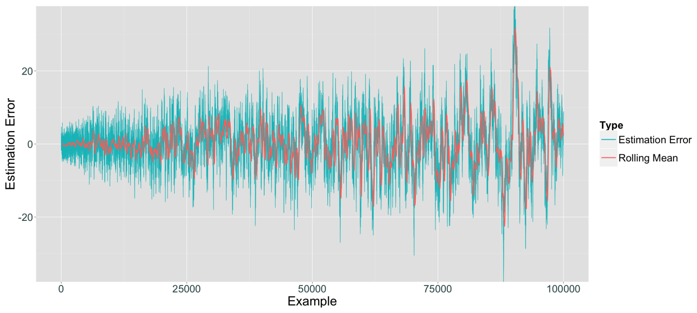


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

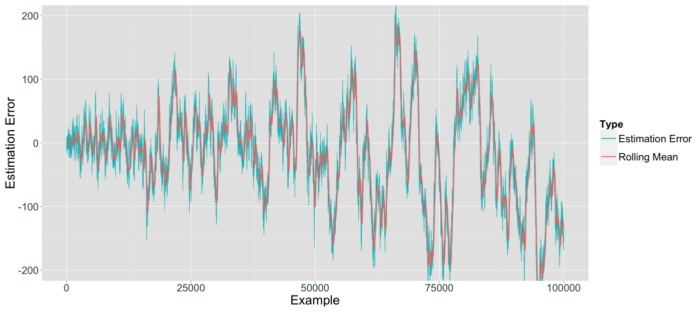
Figure 21: Exp. D3: Response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. It is clear that the adaptability of the estimates strongly decreases over the course of the examples. The accuracy of the estimates deteriorates much faster than for Exp. D1 and D2 in Figures 19 and 19. Where the signal to noise ratio is $R = 10$.



(a) Exp. D1: Estimation errors with rolling mean curves. $R = 0.1$.

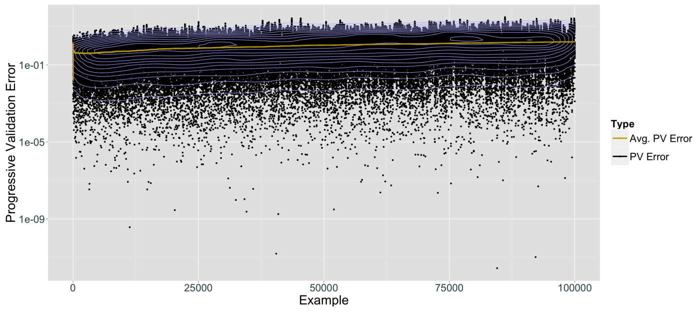


(b) Exp. D2: Estimation errors with rolling mean curves. $R = 1$.

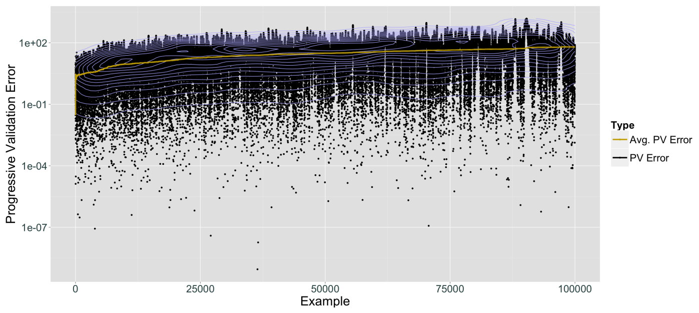


(c) Exp. D3: Estimation errors with rolling mean curves. $R = 10$.

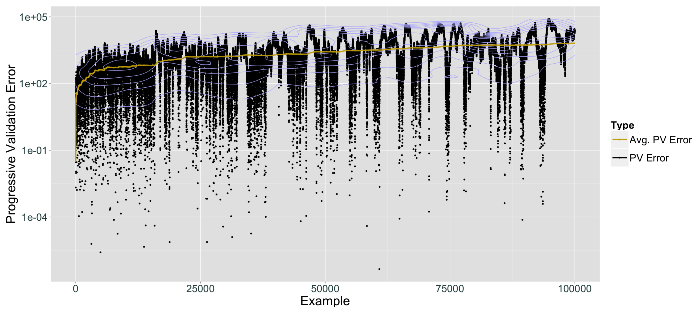
Figure 22: Estimation errors for the tested values of the signal to noise ratio, R. There is evidence of slight error increase and thereby model deterioration for Exp. D1 in Figure 22a, as demonstrated by the slight amplification of the error and rolling mean fluctuations. Exp. D2 and D3 have larger values of R and clearly show increasing error trends, which implies that accuracy of the model estimates deteriorate. This should not be surprising as the learning rates are reduced at steady paces while the model weights continue to fluctuate with the same variance, σ_v^2.



(a) Exp. D1: Pointwise and average progressive validation errors. $R = 0.1$.

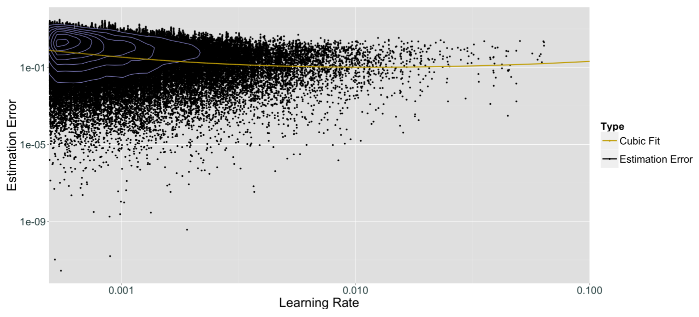


(b) Exp. D2: Pointwise and average progressive validation errors. $R = 1$.

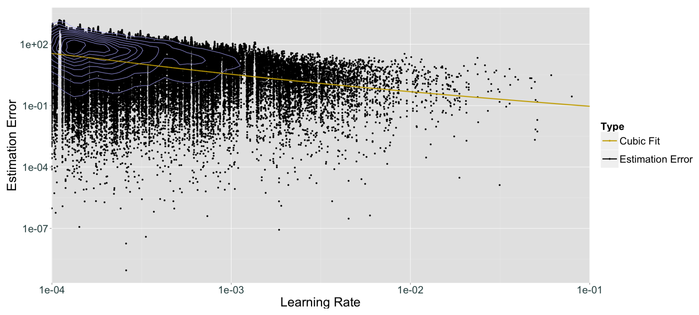


(c) Exp. D3: Pointwise and average progressive validation errors. $R = 10$.

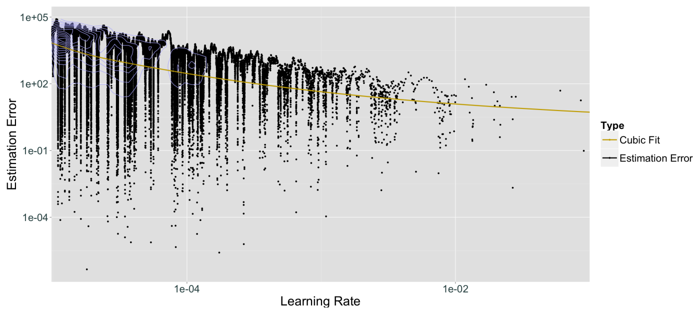
Figure 23: Pointwise and average progressive validation errors for the tested values of signal to noise ratio, R. For Exp. D1 in Figure 23a the progressive validation error trend seems to be quite flat, while Exp. D2 and D3 see clear error deterioration. The plots therefore illustrate on the one hand the model deterioration as R is increased, and on the other that this can be detected by the progressive validation error. The blue contour lines indicate the density of the pointwise progressive validation errors.



(a) Exp. D1: Estimation errors against learning rates. $R = 0.1$

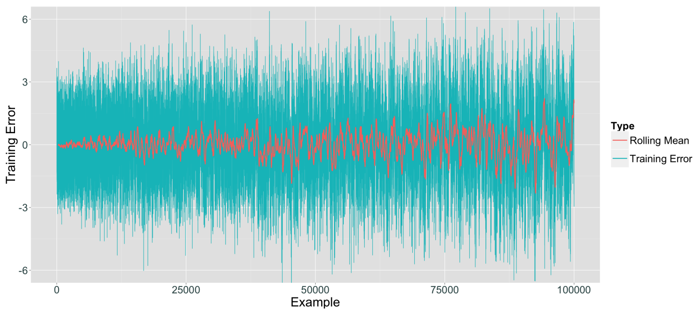


(b) Exp. D2: Estimation errors against learning rates. $R = 1$

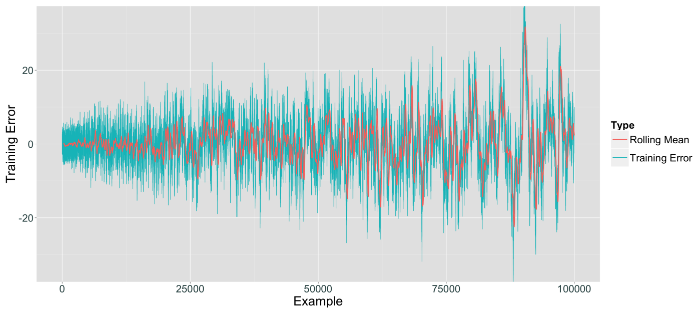


(c) Exp. D3: Estimation errors against learning rates. $R = 10$

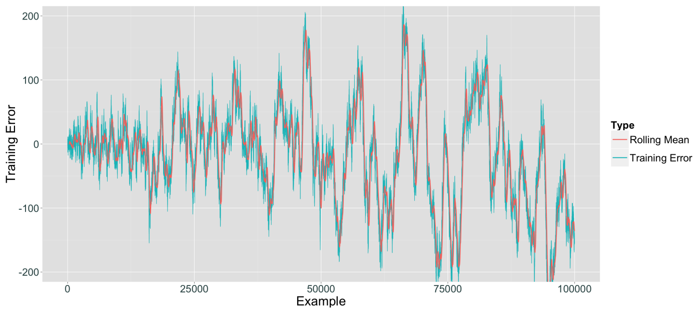
Figure 24: Cubic fits of estimation errors against learning rates indicate negative correlation for all tested values of the signal to noise ratio, R. This suggests that using the AdaGrad implementation, which gradually decreases learning rates, for online learning on nonstationary data, may yield deteriorating model weight estimates. The blue contour lines indicate the point density of the pointwise progressive validation errors.



(a) Exp. D1: Training errors with a rolling mean curve.

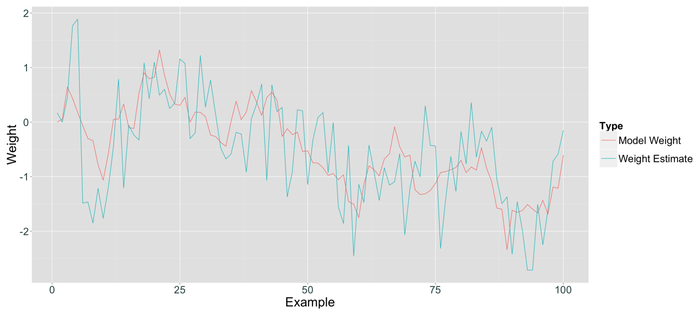


(b) Exp. D2: Training errors with a rolling mean curve.

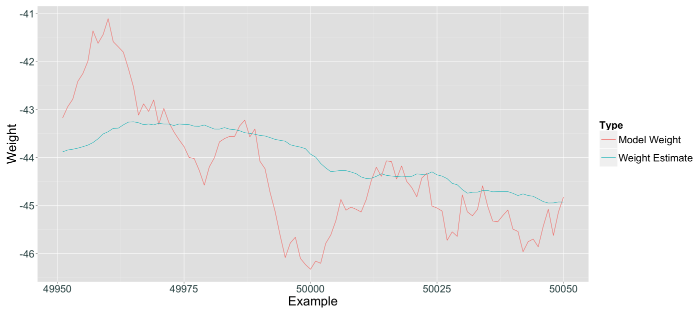


(c) Exp. D3: Training errors with a rolling mean curve.

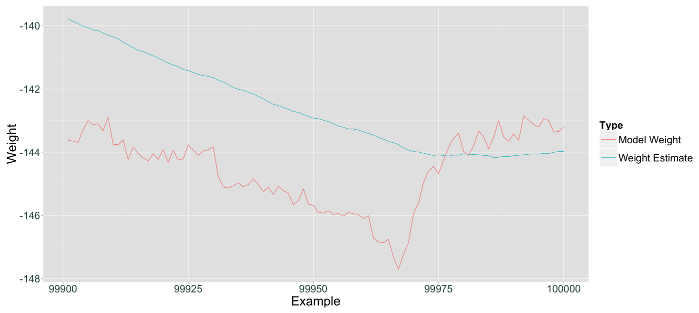
Figure 25: Training errors for the tested values of the signal to noise ratio, R. There is a slight error increase and thereby model deterioration for Exp. D1 in Figure 25a. Exp. D2 and D3 have larger values of R and show clearer error increases, implying that the accuracy of the model estimates deteriorate more. These plots confirm that it is possible to detect model deterioration from the training errors. Note that the training and estimation errors become more similar for increasing R because the variance of the noise, σ_e, then dominates the variance of the signal, σ_v.



(a) Exp. D1: Model weights and weight estimates for the first 100 examples.

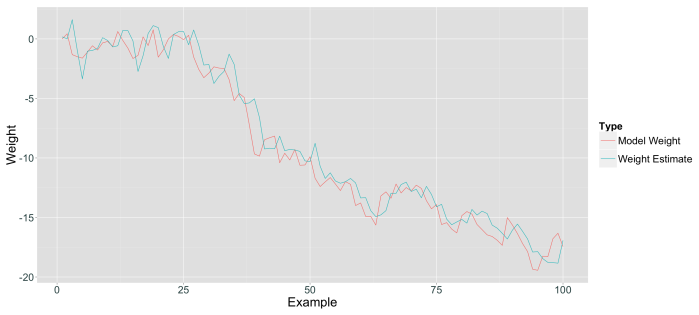


(b) Exp. D1: Model weights and weight estimates for the 100 examples in the middle.

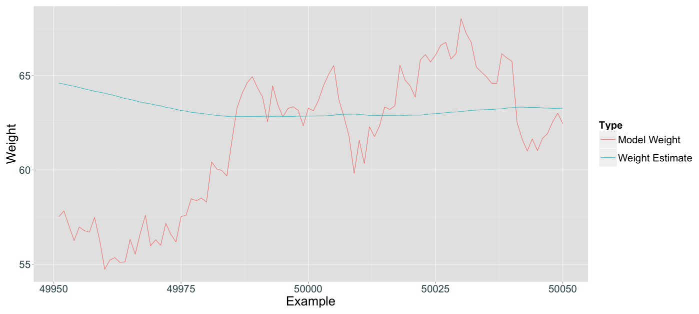


(c) Exp. D1: Model weights and weight estimates for the last 100 examples.

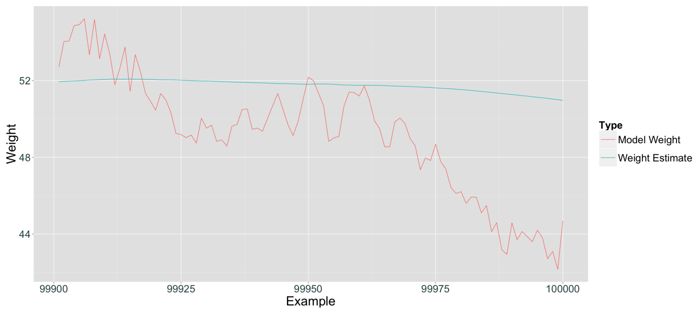
Figure 26: Exp. D.1: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. It is clear that the adaptability of the weight estimates strongly decreases over the course of the examples and become increasingly inaccurate. The signal to noise ratio is $R = 0.1$.



(a) Exp. D2: Model weights and weight estimates for the first 100 examples.

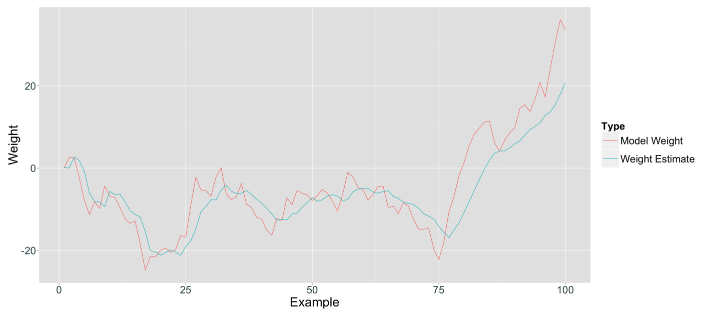


(b) Exp. D2: Model weights and weight estimates for the 100 examples in the middle.

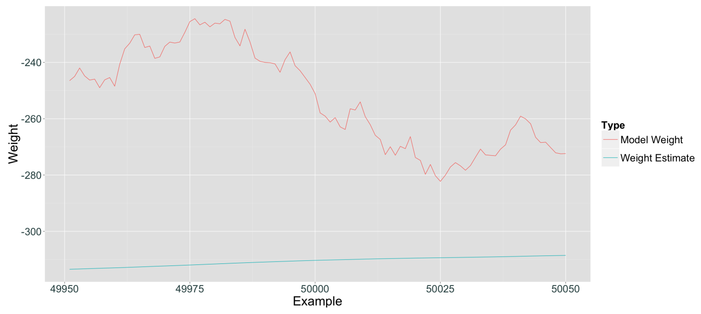


(c) Exp. D2: Model weights and weight estimates for the last 100 examples.

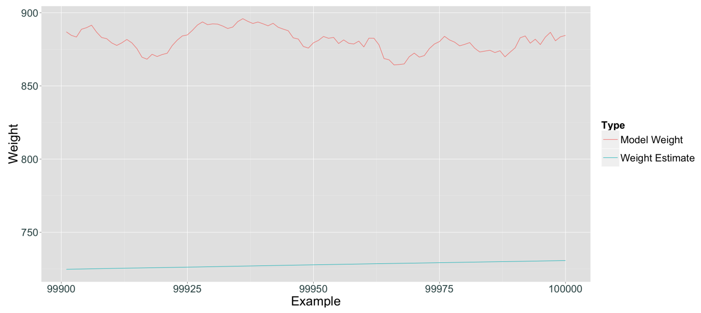
Figure 27: Exp. D.2: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The adaptability of the weight estimates decreases strongly over the course of the examples and become increasingly inaccurate. The signal to noise ratio is $R = 1$. Increasing the signal to noise ratio ten-fold compared to Exp. D1 in Figure 26 had a deteriorating effect on the model weight estimates.



(a) Exp. D3: Model weights and weight estimates for the first 100 examples.

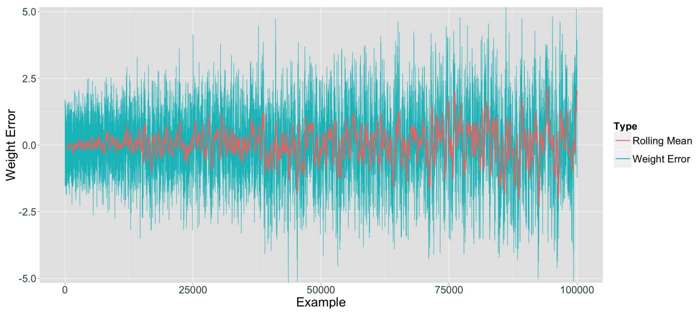


(b) Exp. D3: Model weights and weight estimates for the 100 examples in the middle.

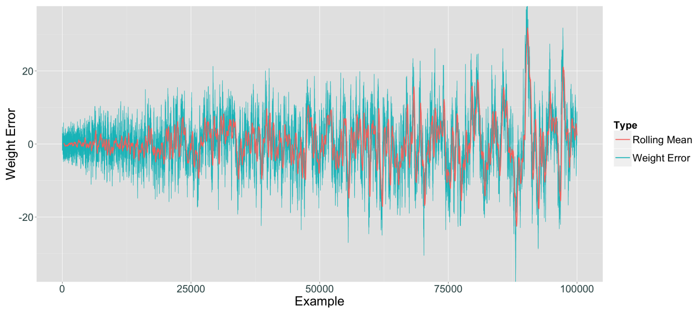


(c) Exp. D3: Model weights and weight estimates for the last 100 examples.

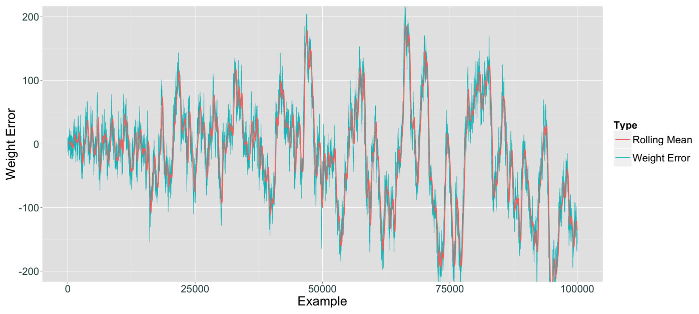
Figure 28: Exp. D.3: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The adaptability of the weight estimates strongly decreases over the course of the examples and become increasingly inaccurate. The signal to noise ratio is $R = 10$. Increasing the signal to noise ratio ten-fold compared to Exp. D2 in Figure 27 had a deteriorating effect on the model weight estimates.



(a) Exp. D1: Weight estimation errors.



(b) Exp. D2: Weight estimation errors.



(c) Exp. D3: Weight estimation errors.

Figure 29: Weight estimation errors for the tested values of the signal to noise ratio, R. There is evidence of slight error increase and thereby model deterioration for Exp. D1 in Figure 29a, as demonstrated by the slight amplification of the error and rolling mean fluctuations. Exp. D2 and D3 have larger values of R and clearly show increasing error trends, which implies that accuracy of the model estimates further deteriorate. This should not be surprising as the learning rates are reduced at steady paces while the model weights continue to fluctuate with the same variance, σ_v^2.

5.2.2 Fixed Learning Rates on Time Series Models

The results of these experiments, described in section 4.2.2, are communicated primarily visually in the plots from Figures 30 to 36. Error plots are added in the appendix[35] for completion, but are omitted here as they are not particularly helpful for interpreting the experimental results.

The primary problem with approaching nonstationary data in an online setting with a fixed learning rate is that there is not necessarily any one learning rate that will perform well on the whole data set. If the learning rate is too low one ends up with underfitting the model to the data, which manifests itself in response estimates and weight estimates that are very slow to adapt to incoming data. This scenario describes the outcome of Exp. D4, as can be seen from Figures 31 and 34.

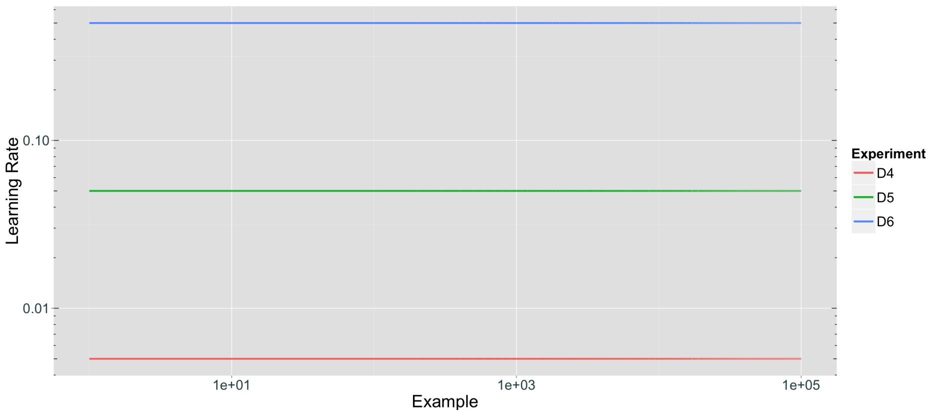


Figure 30: Learning rates for experiments D4-D6 fixed at $\alpha = 0.005, 0.05$ and 0.5 respectively.

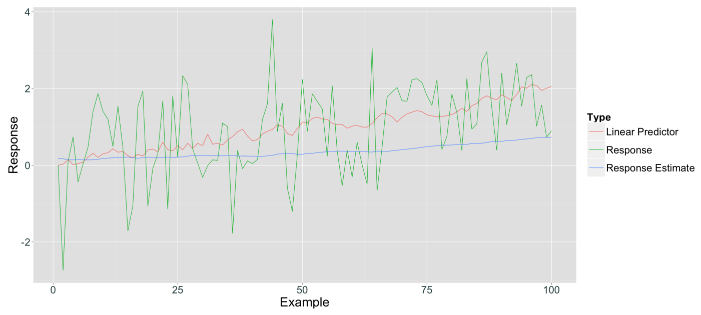
[35]See Figures 49-52 in Appendix C.

On the other hand, setting a too high learning rate yields model overfitting. This results in estimates that follow the observed responses very closely and miss the unobserved linear predictors. The estimated model weights therefore end up missing the generative model weights. This scenario describes the outcome of Exp. D6, as can be seen from Figures 33 and 36.

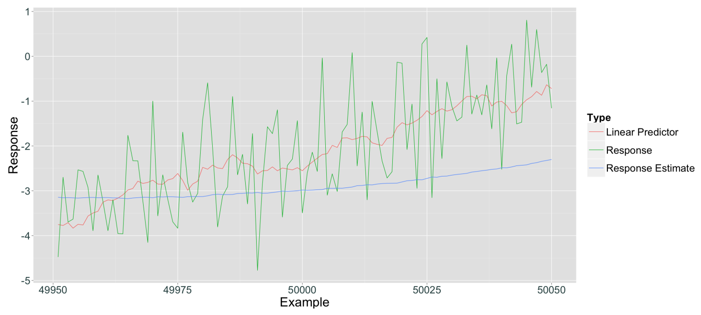
If one manages to select a suitable fixed learning rate then the performance in a nonstationary setting can be very good on parts of the data. For optimally chosen learning rates the estimates filter out the noise of the responses such that they end up following the unobserved linear predictor. And by extension the weight estimates end up following the model weights. The outcome of Exp. D5 is quite promising in this regard, as can be seen from the plots in Figures 32 and 35.

There are two primary weaknesses with this method. The first is that it is not necessarily evident how one selects a fixed step size in an online setting. optimising for fixed step sizes on a nonstationary data set in an offline (batch) setting is far easier where Cross Validation for example is a simple way of selecting the step size while controlling for overfitting. Secondly, if the nonstationary data is generated by a model that changes important properties such as signal to noise ratios or between different kinds of models, then a fixed step size will at best work on parts of the data.

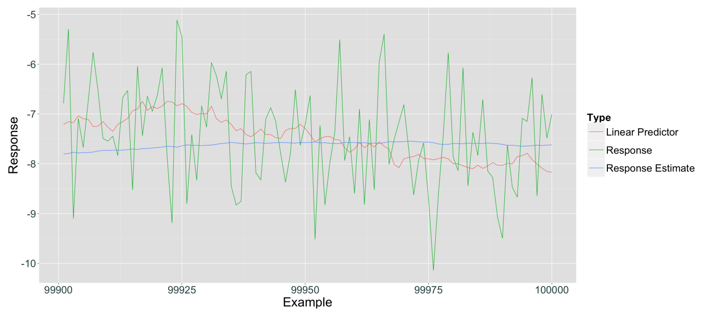
The described limitations of using AdaGrad or fixed learning rates for performing online learning on nonstationary data calls for a more flexible learning method. This method should be able to handle both stationary and nonstationary data that changes character with time, as this is what faces many statisticians working on unsimulated data sets. We propose such a method in the next section and discuss the results of a proof of concept implementation.



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

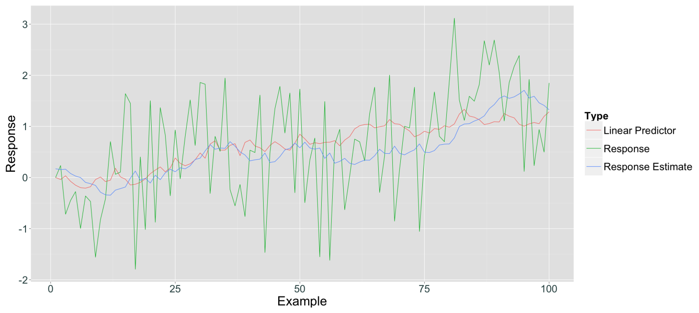


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

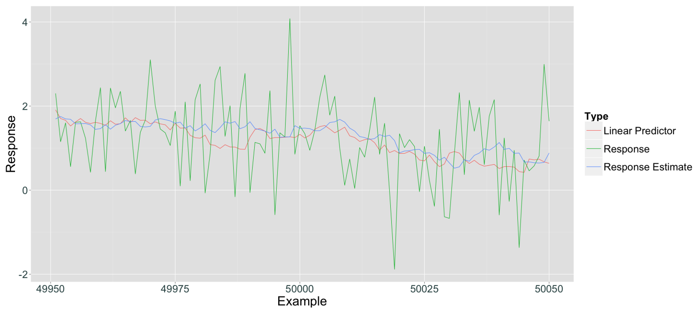


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

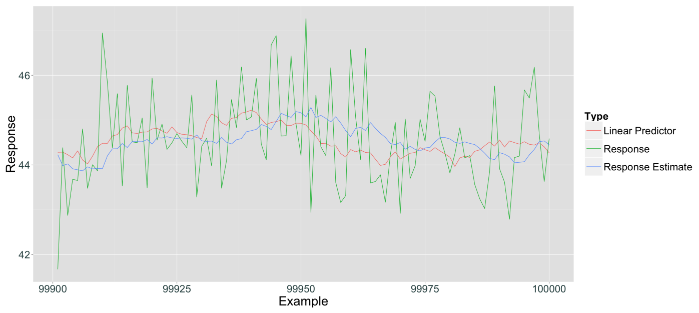
Figure 31: Exp. D4: Response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The estimates adapt poorly to fluctuating responses throughout the data set and miss the fluctuations in the linear predictors. This is an example of model underfitting due to the learning rate being too low for the data set at hand. Increasing it should improve performance. The learning rate is $\alpha = 0.005$.



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

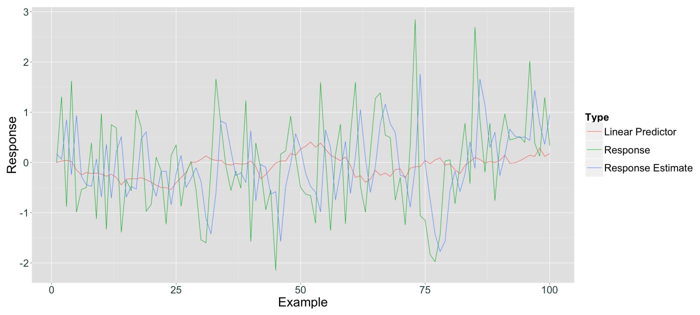


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

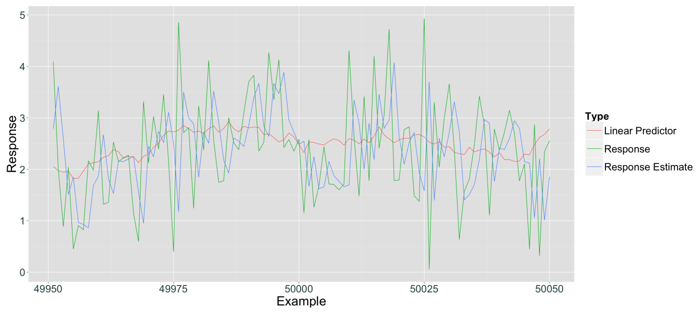


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

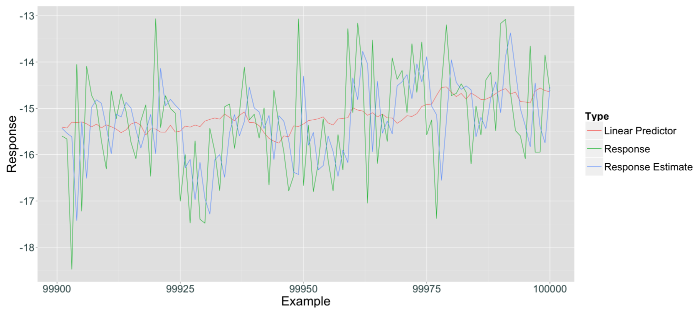
Figure 32: Exp. D5: Response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The estimates follow the linear predictors reasonably well and filter out the noise in the observed responses. This suggests that this learning rate, $\alpha = 0.05$, is close to the optimal.



(a) Response estimates, linear predictors, and observed responses for the first 100 examples.

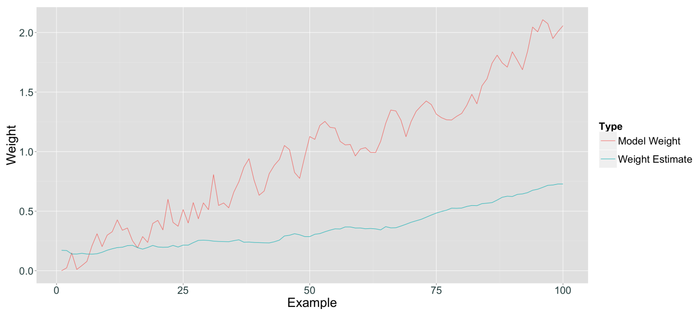


(b) Response estimates, linear predictors, and observed responses for the 100 examples in the middle.

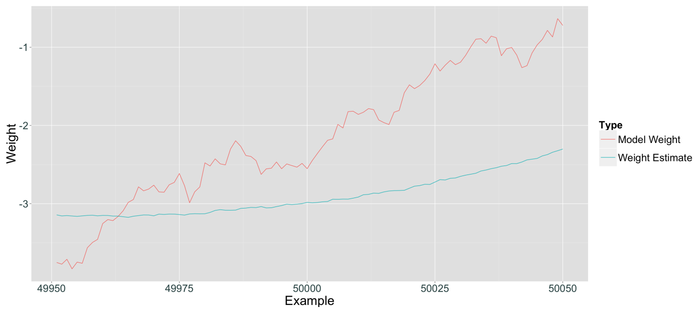


(c) Response estimates, linear predictors, and observed responses for the last 100 examples.

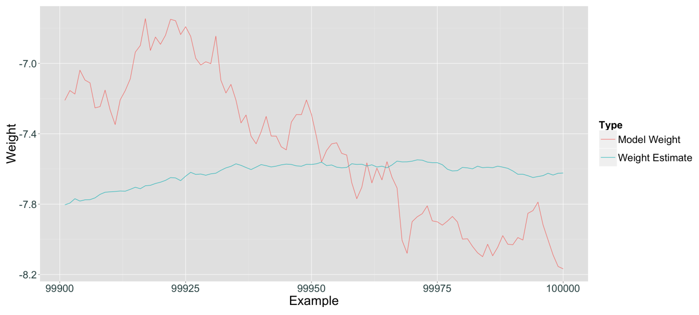
Figure 33: Exp. D6: Response estimates, linear predictors and observed responses for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The estimates are too adaptive to the fluctuations of the responses throughout the data set as they follow the responses extremely closely, but miss the linear predictors. This constitutes model overfitting and is caused by a too high learning rate. The learning rate is $\alpha = 0.5$.



(a) Exp. D4: Model weights and weight estimates for the first 100 examples.

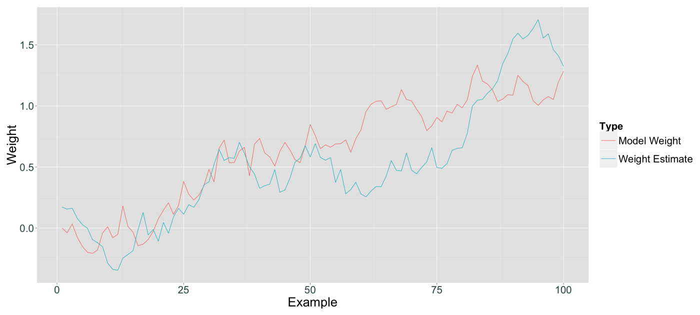


(b) Exp. D4: Model weights and weight estimates for the 100 examples in the middle.

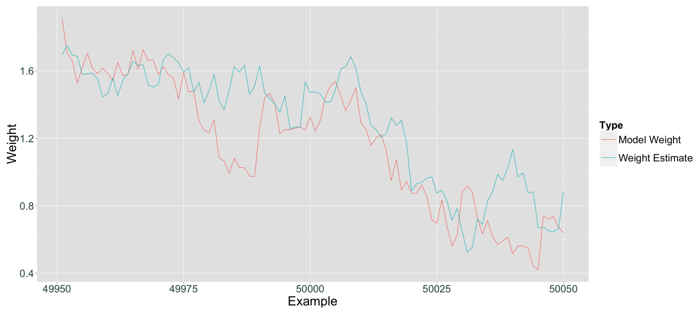


(c) Exp. D4: Model weights and weight estimates for the last 100 examples.

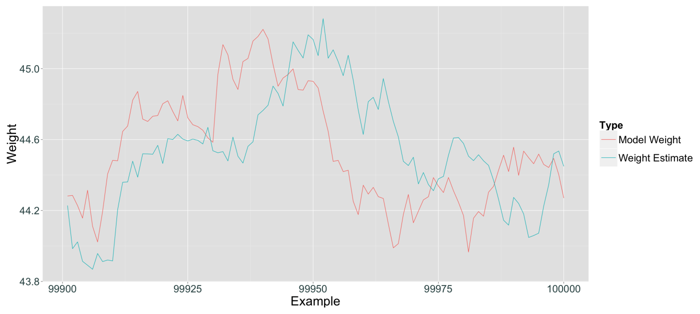
Figure 34: Exp. D4: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. Throughout the data set the weight estimates adapt poorly to the fluctuating responses and miss the generative model weights. This is an example of model underfitting due to the learning rate being too low for the data set at hand. Increasing it should improve performance. The learning rate is $\alpha = 0.005$.



(a) Exp. D5: Model weights and weight estimates for the first 100 examples.

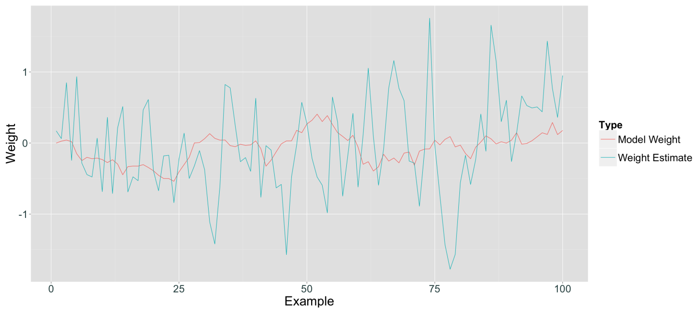


(b) Exp. D5: Model weights and weight estimates for the 100 examples in the middle.

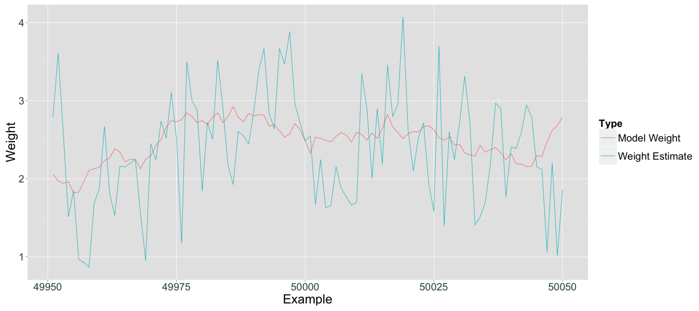


(c) Exp. D5: Model weights and weight estimates for the last 100 examples.

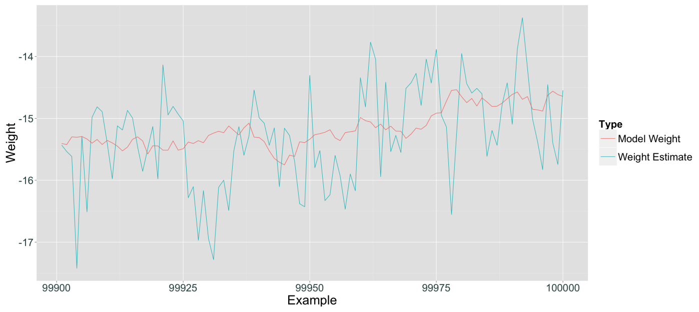
Figure 35: Exp. D.5: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. The weight estimates adapt reasonably well to the fluctuating responses throughout the data set. The weight estimates follow the model weights reasonably well and filter out the noise in the observed responses. This suggests that this learning rate, $\alpha = 0.05$, is close to the optimal.



(a) Exp. D6: Model weights and weight estimates for the first 100 examples.



(b) Exp. D6: Model weights and weight estimates for the 100 examples in the middle.



(c) Exp. D6: Model weights and weight estimates for the last 100 examples.

Figure 36: Exp. D6: Model weights and weight estimates for the 100 examples at the start, middle and end of the data set, where the number of data points is $n = 10^5$. Figure 33 revealed that the estimates are too adaptive to the fluctuations of the responses for this learning rate, $\alpha = 0.5$. This translates to excessively fluctuating weight estimates that fail to estimate the generative model weights and is an example of overfitting due to too high α.

6 PSGD

6.1 Motivation

The results in section 5.2 show that the performance of `vw` can seriously deteriorate over time in an online setting with nonstationary data. The AdaGrad method causes learning rates to be strictly decreasing, making it suitable for model fitting on many stationary data sets, but unsuitable for performing online learning on extremely large nonstationary data sets. This is due to a decreasing adaptiveness of the response estimates and weight estimates as learning rates decrease. Alternatively, one may set a fixed learning rate, but as the results in section 5.2.2 indicate, this method is prone to underfit or overfit the data if the generative model changes with time.

There is a class of solutions to this problem that we will not consider. This includes solutions customized to the generative models of the data sets. These solutions minimize model misspecification between the generative model and the estimated model as much as possible. Though this approach can yield good results, customized solutions are impractical in an online setting where one knows very little about the generative model of the incoming data and are required to deliver a high volume of predictions on the fly.

Relatively little attention has been devoted to the study of online learning in nonstationary settings in the machine learning literature. Due to time constraints a thorough literature review of methods addressing this problem has not been possible. However, recently proposed second-order methods to solve the problem of online learning in nonstationary settings are criticised for being too computationally intensive, as they either require matrix inversion or eigenvalue decomposition [25].

We address this problem by propose framework in which several SGD learners are run in parallel for each data point. The cost of this class of methods is that it requires more processing power and memory than other SGD variants. However, these costs are relatively small, as SGD is already very computationally efficient, and would therefore not impede the scalability of the proposed method. The results in section 6.4 are so promising for this class of problems that successful further research on the discussed weaknesses can motivate a larger scale implementation, for example in `vw`.

6.2 Method

Consider a realistic online learning setting where data is continuously arriving and the unknown models generating this data change with time.

For each data point, the proposed approach consists of running 3 SGD learners with different step sizes, α, in parallell[36]. One learner is started at the initial learning rate $\alpha_c = \alpha_i$, and is termed the current learner. While the other two have learning rates that are scaled up and down by a given constant parameter $S > 1$, such that $\alpha_u = S\alpha_c$ and $\alpha_d = \alpha_c/S$. We call these the upper and lower learners.

Upon the arrival of each data point the gradient of the loss is computed. The model weights are subsequently estimated for each example, k, according to equation (6.1)[37]

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha_k \nabla_{\boldsymbol{\beta}} L(\boldsymbol{z}_i, \widehat{\boldsymbol{z}}_i | \boldsymbol{\beta}_k). \tag{6.1}$$

Note that for each data point the gradient only needs to be computed once as it is the same for all three learners. This is because the loss is based on the previous estimate of the current learner for each data point. The only difference in the weight estimation from equation (6.1) between the three learners is therefore the value of α_k. This makes the parallellization more efficient and scalable.

[36]Though 3 is the minimum number of learners necessary to follow the principles of our proposed PSGD method, there is nothing preventing us from running more learners in parallel. This is however not necessary to obtain good results.

[37]Introduced as equation (3.1) in section 3.

The mean of the one-step ahead prediction error, $\bar{\epsilon}$, a proxy of the generalization error, and the standard deviation of the errors, s_ϵ, can be sequentially updated for each data point, for each of the processes, according to

$$s_n^2 = \tfrac{n-2}{n-1} s_{n-1}^2 + \tfrac{(x_n - \bar{x}_{n-1})^2}{n} \tag{6.2}$$

$$\bar{x}_n = \tfrac{(n-1)\bar{x}_{n-1} + x_n}{n} \tag{6.3}$$

or by using other algorithms that are less prone to numerical instability, but which may be less efficient, such as the method discussed by Knuth (1998). The mean and standard deviations of the one-step ahead prediciton errors of the processes are monitored in order to assess whether or not the upper or lower learners have mean errors that are statistically significant smaller than the mean errors of the chosen learner. The learning rate of the current learner is changed to that of the upper or lower learning rate if one of the following inequalities are satisfied

$$\bar{\epsilon}_d - \bar{\epsilon}_c < -z\sqrt{s_{\bar{\epsilon}_d}^2 / n_\epsilon + s_{\bar{\epsilon}_c}^2 / n_\epsilon}\ , \qquad \text{reduce } \alpha \tag{6.4}$$

$$\bar{\epsilon}_u - \bar{\epsilon}_c < -z\sqrt{s_{\bar{\epsilon}_u}^2 / n_\epsilon + s_{\bar{\epsilon}_c}^2 / n_\epsilon}\ , \qquad \text{increase } \alpha \tag{6.5}$$

where z, typically 1.96, determines the size of the one-sided rejection regions and n_ϵ is the number of examples used to update the estimates of the mean and standard deviations. Note that after a switch is made the estimates of the means and standard deviations are reset such that the effect of errors from a potentially old regime do not influence the error means and standard deviations going forward. Bear in mind that the mean one-step ahead prediction error is in fact the progressive validation error, or PV error, described in section 4.3.

The inequalities in equations (6.4) and (6.5) are motivated by the assumption that if the one-step ahead prediction errors are independent, then according to the Central Limit theorem [5] means are asymptotically normal, from which the rejection regions in the form of inequalities can be derived. This assumption does not necessarily hold, motivating the derivation of more precise results. However, a proof-of-concept implementation of the method with this approximative switching rule, in the one parametric case, yields promising results.

This parallelised approach is designed specifically for nonstationary settings and should, with appropriate tuning of S, and z, by definition be able to yield results with smaller PV errors in nonstationary online settings than the AdaGrad and fixed learning rate methods considered previously. This is confirmed in section 6.4. However, this enhanced performance comes at a cost. The first is that the method requires more processing power and memory since it is in essence running SGD 3 times per data point. However, due to the possibility of parallelising the method, this should not significantly increase the running time. Note that only needing to compute the gradient once per data point makes the implementation even more computationally efficient, and that the increased memory requirement is due to storing the PV error and its standard deviation for the three processes. Given that the memory requirements of online SGD is already very low, since only one data point is processed and stored in memory at any given time, this should not affect the scalability of the method. The sequential checks for whether the learning rates should be scaled up or down are also computationally cheap and scalable.

The performance of the proposed implementation of PSGD, just as SGD, is sensitive to the learning rate parameter, α. Experience shows that it is seldom warranted to initialise a learning rate at 1 or above, as this can cause SGD to either yield extreme overfitting or estimate divergence, and one might therefore also implement a safety mechanism preventing PSGD of increasing α beyond $\alpha_{\max} = 1$. Experience also shows that it is rarely necessary to have learning rates as low as $10^{-8} < \alpha < 10^{-6}$. So implementing a minimum learning rate value, $\alpha_{\min}$, might also be considered. Note that in none of the following experiments have the PSGD learning rates reached any predefined thresholds, so this has not influenced the results or conclusions in any way.

6.3 Design of Experiments

The purpose of the experiments in this section is twofold. Firstly, they seek to compare the performance of PSGD with the AdaGrad and fixed learning rate implementations of SGD in `vw` on some of the experiments discussed in section 5.2. Secondly, and perhaps more importantly, they seek to shed light on the properties of the proposed PSGD method. The ultimate goal of the proposed learning method is to be more robust in the face of realistic nonstationary data streams, than what the online learning methods implemented in `vw` are today. However, the data sets considered in this section are only meant as informative test scenarios, and not attempts to accurately mimick realistic data streams.

Note that comparing our PSGD method with AdaGrad can be criticised as unfair since we have shown and explained that AdaGrad is clearly not designed for handling nonstationary data. The comparison is however included by merits of being a benchmark and because a key developer of `vw` has expressed the belief that AdaGrad can actually handle nonstationary data[38] - a misconception that therefore is unfortunately most likely circulating among some data scientists.

Three different kinds of experiments will be shown.

The first experiment is described in Table 3 of section 4.2.2 with the name D1. This experiment is chosen as it is the most noisy of these time series experiments, permitting us to assess the extent to which PSGD overfits a model to the data while comparing its performance with that

[38]From the most upvoted answer on a Stack Overflow question: "Online learning is adaptive and can track changes in conditions over time, so it can learn from non-stationary data, like learning against an adaptive adversary." `http://stackoverflow.com/questions/24822288/correctness-of-logistic-regression-in-vowpal-wabbit`

of vw. Note that the PSGD learning rate will be set to 1, which is close to the initial learning rate of of AdaGrad in vw[39].

The second experiment is described in Table 4 of section 4.2.2 with the name D5. This experiment is chosen as it is the one with the seemingly most suitable fixed learning rate for the given problem. Note that the PSGD learning rate will be started from the same learning rate as the fixed learning rate process.

In both the first and second experiment we set the scaling factor, $S = 1.5$, and standard score, $z = 1.96$, in equations (6.4) and (6.5)

Lastly the performance of PSGD is assessed on data sets that are a mixture of different stationary and nonstationary data in order to further examine its adaptiveness. As the current proposed implementation of PSGD does not ideally adapt the learning rates in all settings, the focus will be on scaling up and down the signal to noise ratios, R, for different segments of the incoming data in a controlled fashion. Three scenarios are considered in section 6.4 and are summarized in Table 5.

Note that, as discussed in section 4, avoiding model misspecification between the training model and the generative model should not be a goal in an online setting. The model used to fit the stream of observed responses y_t by the proof-of-concept implementation of PSGD is a simple linear model with only a intercept. Despite the obvious model misspecification the performance of PSGD is promising.

If time had allowed it, these experiments could have included comparisons to the second-order online methods proposed by Vaits et al. (2015), but which are more computationally intensive than our proposed PSGD approach.

[39]The reason the learning rates of AdaGrad were not accurately initialised to 1 is that AdaGrad adapts the learning rates to the incoming data.

Table 5: Parameter settings for three PSGD experiments examining the learning rate adaptiveness as the signal to noise ratio, R, and thereby the response variance, is varied. Note that n examples per value of R are simulated and that the standard deviation of the signal process, σ_v, is held constant. The learning rate is initialised at α_i. The upper and lower learners are subjected to scaling factor, S, and are sequentially tested against the current learner to identify the learner with the statistically significant smallest progressive validation error. For these tests the standard score, z, is used.

Experiment	R	σ_v	α_i	S	z	n
M1	(0.1,1,10)	1	0.4	1.5	1.96	10^5
M2	(10,1,0.1)	1	0.4	1.5	1.96	10^5
M3	(0.1,10,0.1)	1	0.1	1.5	1.96	10^5

6.4 Results

This section contains the results of the experiments described in the previous section. Note that most learning rate initialisations for PSGD yield the same conclusions as discussed below. However, for too high learning rate initialisations, from $\alpha \approx 1$ depending on the data, the errors can diverge. This has never been seen to be accompanied by PSGD scaling up or down the learning rates. The error divergence can therefore be attributed to SGD causing diverging estimates at elevated learning rates, and not by a weakness in the SGD extension in the proposed PSGD method. However, ideally PSGD would pick up on the excessively large learning rate and reduce it. This can be done by increasing the scaling factor of the upper and lower learners. This does however come at a cost, as it reduces the resolution of the learning rates PSGD considers. An adaptation to this could be to either set the initial learning rate more carefully, at most around 0.5, or to have two or more upper and lower learners with a sufficient span in the scaling factors.

On the other hand, too low learning rate initialisations cause different results if PSGD does not manage to increase the learning rates close enough to the learning rate of the vw benchmark processes. In an online setting, however, this should not be a problem as PSGD should achieve appropriate learning rates with time.

6.4.1 AdaGrad Comparison

This section provides a comparison between the performance of PSGD and AdaGrad on a simple experiment outlined in Table 3 of section 4.2.2 under the name D1. The learning rates of both PSGD and AdaGrad[40] were initialised at $\alpha \approx 1$. It is worth rementioning that AdaGrad is clearly not designed for handling nonstationary data as discussed in section 5.2, but

[40]The reason the learning rates of AdaGrad were not accurately initialised to 1 is that AdaGrad adapts the learning rates to the incoming data.

that the comparison is carried out to address the misconception among some data scientists that AdaGrad can handle nonstationary data.

Both AdaGrad and PSGD adapt its learning rates to the incoming data. In this case this is marked by decreases from the initial value, as seen in Figure 37. PSGD, however, stabilises its learning rates at a point where further changing the learning rates will not cause the PV errors, or average one-step ahead prediction errors, to be statistically significant smaller than the PV errors of the upper and lower learners. The value at which PSGD stabilises the learning rate is considerably higher than that of AdaGrad, and is the fundamental reason why PSGD clearly outperforms AdaGrad in this experiment, in terms of significantly lower cumulative training and estimation errors, as demonstrated by Figure 38.

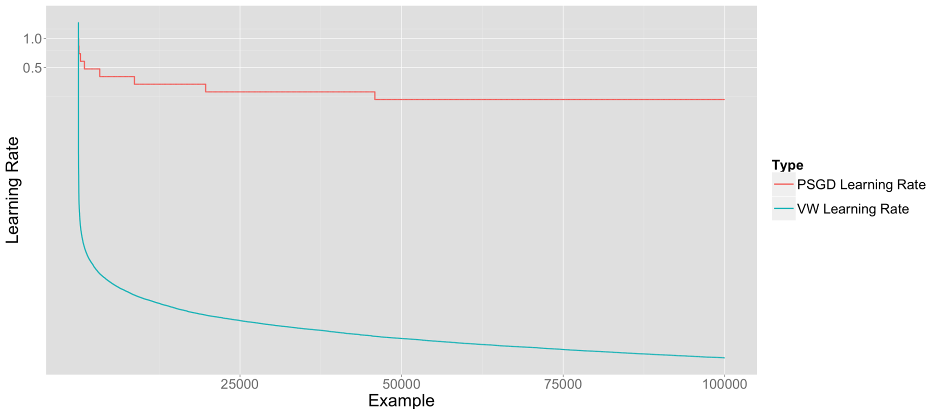
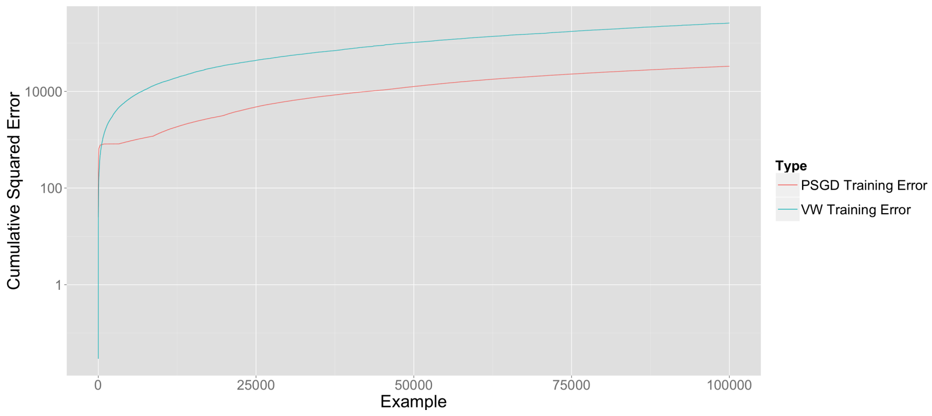
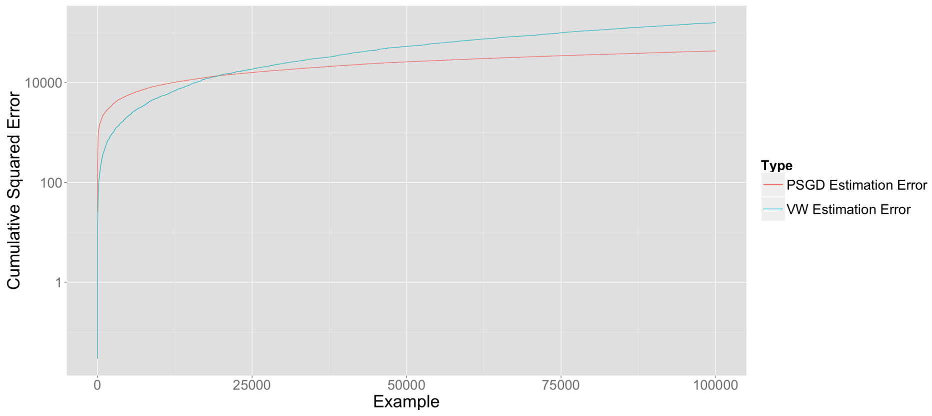


Figure 37: AdaGrad and PSGD learning rates after a run of PSGD on non-stationary data specified by Experiment D1. Note that PSGD adapts to the data gradually by decreasing the learning rates before they stabilise around 0.2. AdaGrad on the other hand decreases the learning rate much faster, despite also adapting to the data.



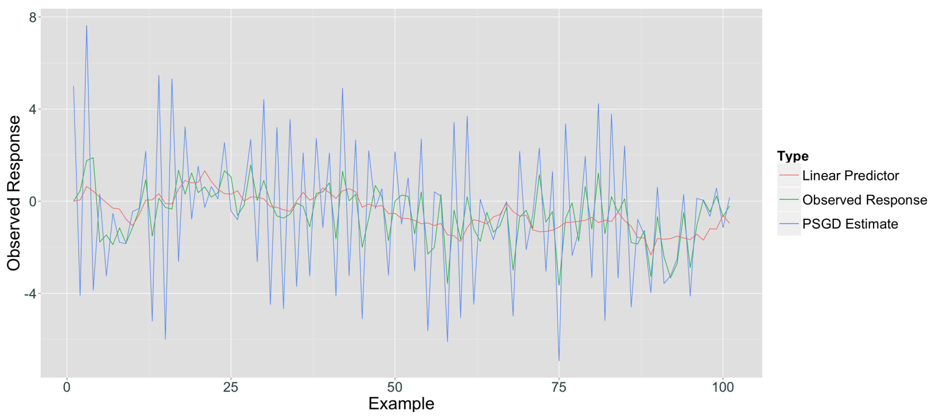
(a) Cumulative training errors of the PSGD and AdaGrad method.



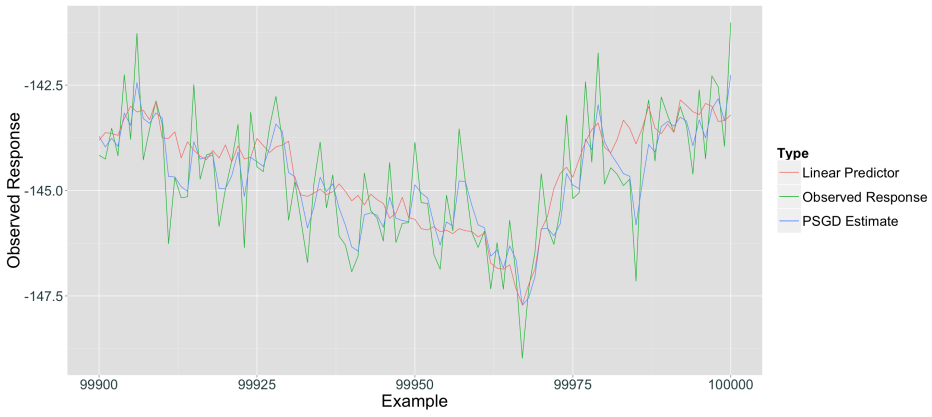
(b) Cumulative estimation errors of the PSGD and AdaGrad method.

Figure 38: The cumulative estimation and training errors of the PSGD method are, over time, clearly smaller than those of the AdaGrad method. This suggests that the PSGD method is more suitable for the nonstationary data of this experiment. Note, however that the estimation errors for the first examples are higher for the PSGD method. This is because AdaGrad in `vw` decreases the learning rates much faster than PSGD, which in this case is beneficial for the first examples, but in general depends on the learning rate initialisation.

A key issue when fitting models to data is to prevent overfitting. In Figure 39 the observed responses, linear predictor and PSGD estimates of the first and last 100 examples, show that the initial learning rate was so high that it caused extreme overfitting, and that the learning rate reduction considerably reduced the extent of overfitting. Another way of examining overfitting is to see if the training error is reduced at the expense of an increased estimation error. Figure 40 indicates that this does not describe the performance of PSGD. For decreasing learning rates, the estimation errors strictly decrease, while the training errors decrease before they increase again.

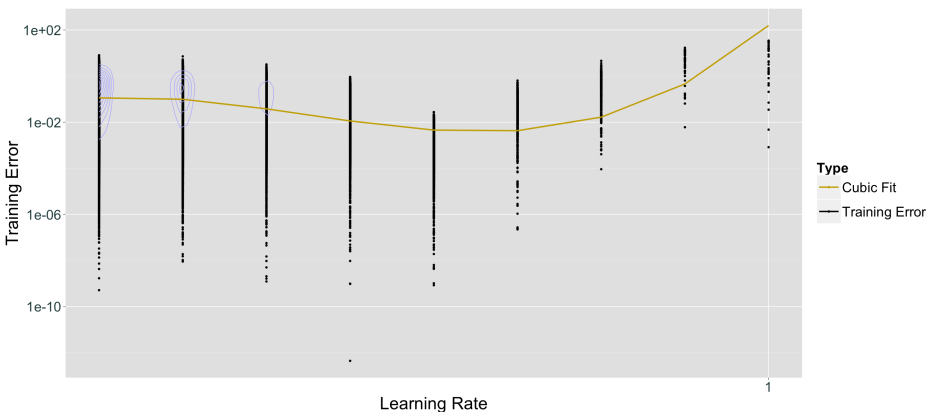


(a) Linear predictor, observed responses and PSGD estimates for the first 100 examples.

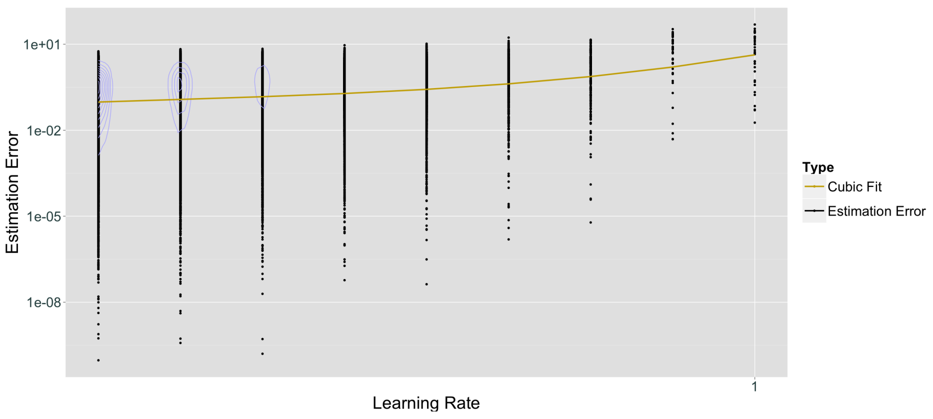


(b) Linear predictor, observed responses and PSGD estimates for the last 100 examples.

Figure 39: Linear predictor, observed responses and PSGD estimates for the first and last 100 examples. It is evident that the learning rate is initialised at a too high value as the volatility of the estimates is a lot higher than the volatility of the responses for the first 100 examples. After the learning rate has been reduced and has stabilised the PSGD estimates match the linear predictor a lot better by partly filtering out the signal from the noise, as can be seen from the last 100 examples.



(a) Training errors plotted against learning rates.



(b) Estimation errors plotted against learning rates.

Figure 40: Training errors and estimation errors are plotted against learning rates from the PSGD method. Note that learning rates are decreased over time as seen from Fig. 37. Estimation errors decrease as learning rates are decreased, while the training errors reach a minimum before increasing again as the learning rate is reduced to around 0.2. This suggests that the adaptiveness of the learning rate reduces the generalization error without blindly overfitting the model to the data.

6.4.2 Fixed Learning Rate Comparison

This section provides a comparison between the performance of PSGD and `vw` with a fixed learning rate on a simple experiment outlined in Table 4 of section 4.2.2 with the name D5. Figure 41 shows that PSGD relatively quickly adapts its learning rate to the incoming data. It is reduced to being around a fifth of its initial value - only twice the value of the fixed learning rate - before it stabilises[41].

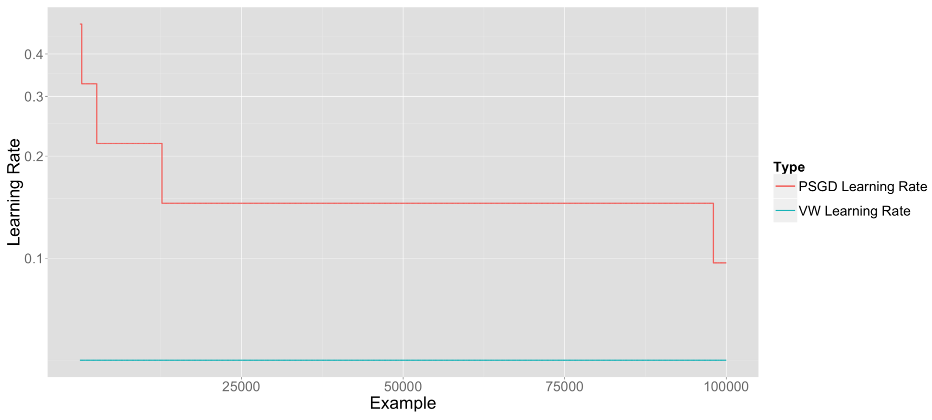


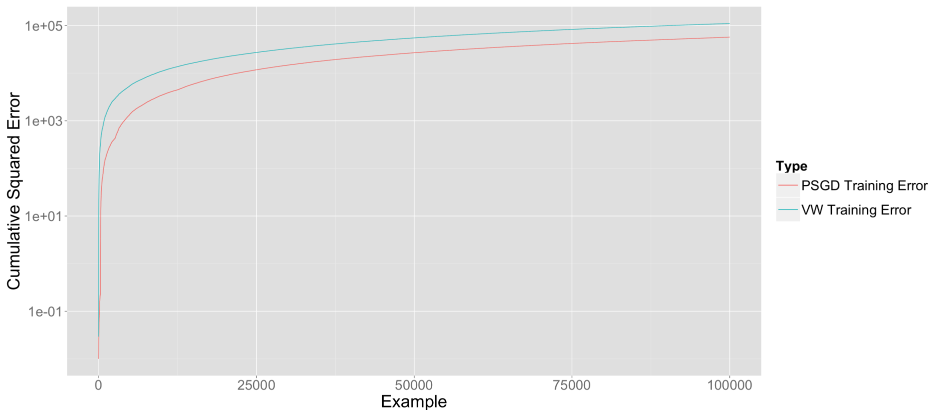
Figure 41: Plot of the fixed learning rate of 0.05, set in `vw`, for experiment D5 and the PSGD learning rates initialised at 0.5, the value from exp. D6. Note that PSGD adapts to the data on the fly by decreasing its learning rate before it stabilises around 0.1. This removes the necessity of performing Cross Validation in an offline setting to select an appropriate fixed learning rate for SGD.

The first examples in Figure 42, during which the learning rate for PSGD is very high and close to the initial value, are riddled with very high errors compared to `vw`. The fast learning rate decrease can be seen to reduce the rate of cumulative error increase for these first examples.

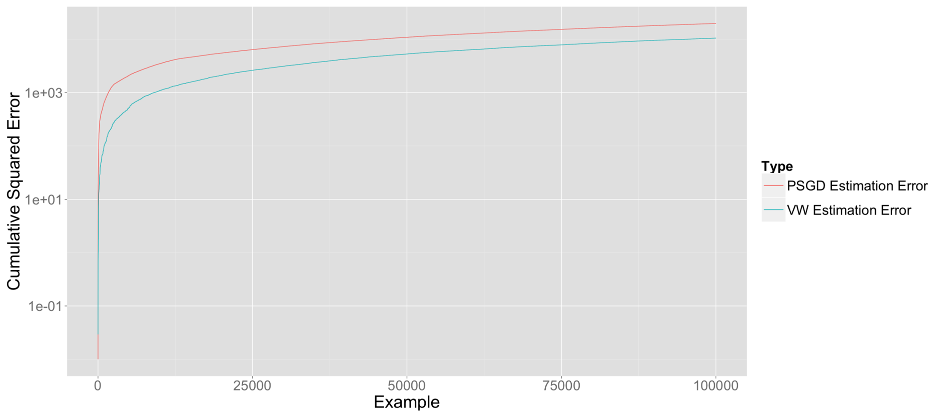
[41]Increasing the number of data points, n, might cause further changes to the learning rate, but this is not important to this discussion.

After the learning rates stabilize the cumulative errors of both PSGD and `vw` show similar trends. So, despite an adverse learning rate initialization, PSGD adapted to the data to perform well compared to a well-initialised fixed learning rate. This suggests that PSGD reduces the necessity of using Cross Validation or other techniques to identify appropriate initial learning rates. This is a necessary feature of a useful online learner in a nonstationary setting.

The observed responses, linear predictors and PSGD estimates for the first and last 100 examples are plotted in Figure 43. They show that the high learning rate initialisation causes considerable overfitting, but that the extent of this is significantly reduced by the last 100 examples, due to the learning rate reduction. Thus, again, the learning rate adaptiveness of the PSGD method does not seem to cause overfitting, rather it is able to prevent it. This is further confirmed by Figure 44 showing that estimation errors decrease as the learning rates decrease, most likely reducing the generalization error of the method.

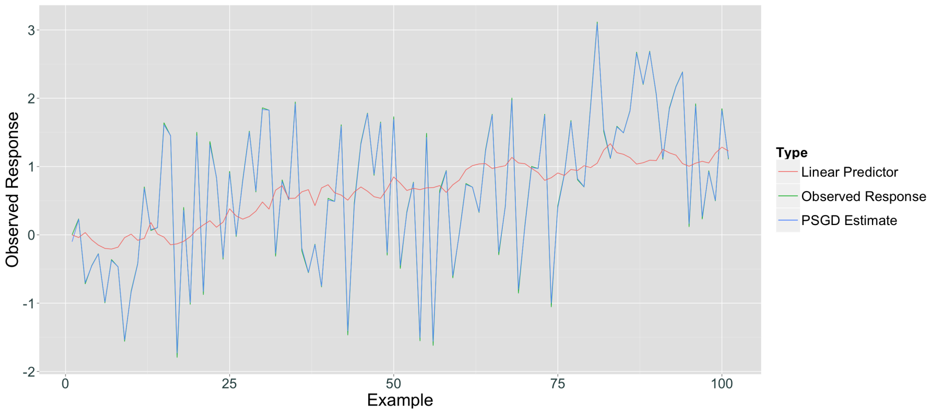


(a) Cumulative training errors of `vw` with fixed learning rates and PSGD.

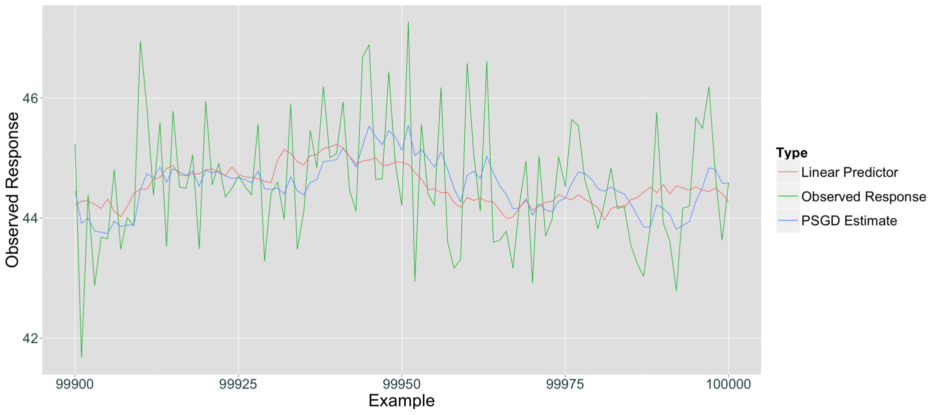


(b) Cumulative estimation errors of `vw` with fixed learning rates and PSGD.

Figure 42: The cumulative estimation and training errors of the PSGD method approach those of `vw` with a fixed learning rate of 0.05, after the learning rate adjustments. This suggests that the PSGD method manages to adapt to the data on the fly, approaching the performance of `vw` with the well-chosen fixed learning rate of 0.05.

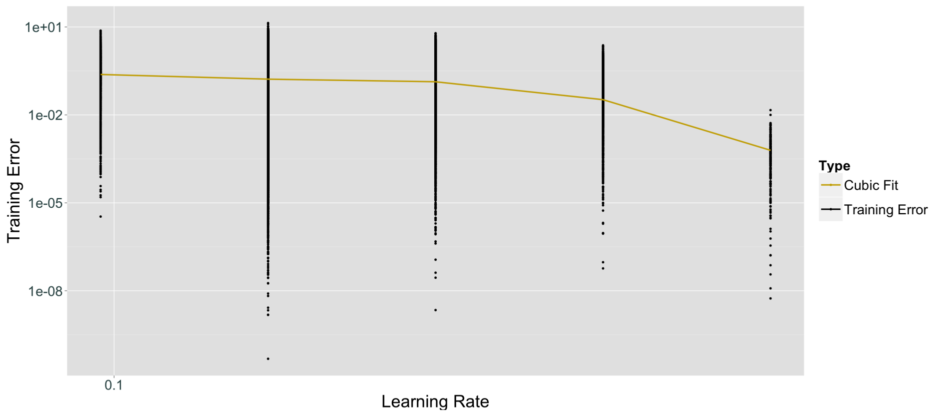


(a) Linear predictor, observed responses and PSGD estimates for the first 100 examples.

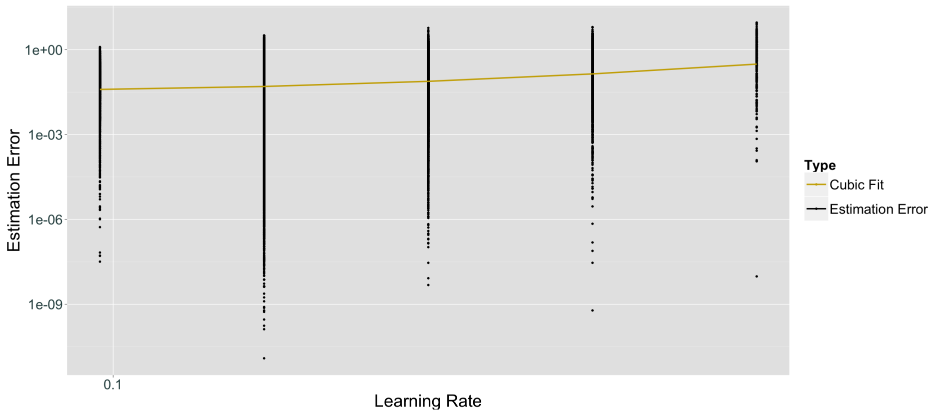


(b) Linear predictor, observed responses and PSGD estimates for the last 100 examples.

Figure 43: Linear predictor, observed responses and PSGD estimates for the first and last 100 examples. It is evident that the learning rate is initialised at a too high value as the estimates are almost completely matching the responses for the first 100 examples, an example of extreme overfitting. Despite this situation producing very small training errors, the PSGD method decreases the learning rates, thereby reducing model overfitting and the estimation errors. This is clear from the last 100 examples of the processed data set.



(a) Training errors of PSGD plotted against learning rates.



(b) Estimation errors of PSGD plotted against learning rates.

Figure 44: Training errors and estimation errors are plotted against learning rates from the PSGD method. Note that learning rates are decreased over time as seen from Fig. 41. Training errors increase as learning rates are reduced, while the opposite is true for estimation errors, the proxy of the generalization error. This suggests that the adaptiveness of the learning rate reduces the generalization error and has not come at the cost of overfitting the model to the data.

6.4.3 Nonstationary Data from Multiple Models

Though the previous sections comparing PSGD against SGD with AdaGrad or with fixed learning rates yield positive results, all the simulations used for those experiments originate from a given generative model that does not change with time. The goal of PSGD, or online learning in a nonstationary setting, should be to be able to adapt the model weight estimates to incoming data originating from different generative models over time. This section discusses the performance of PSGD in such settings.

The experiments of this section are summarized in table 5 of section 6.3. In experiment M1 the signal to noise ratio, R, is scaled up after the first and second third of the examples are simulated. In experiment M2 the signal to noise ratio is scaled down instead. And in experiment M3 the signal to noise ratio is scaled up once before it is scaled down to its initial value.

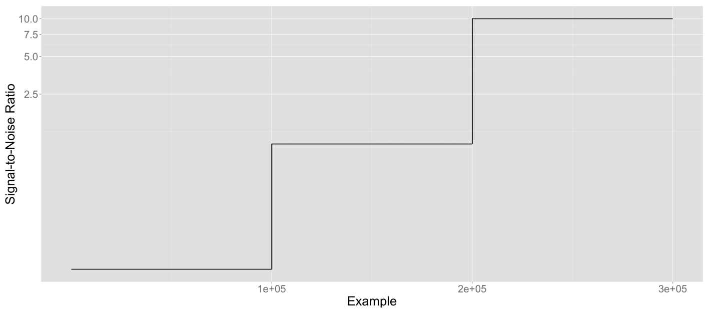
The overall performance of PSGD is superior to `vw` with a fixed learning rate initialised at the same value as PSGD[42] for the described experiments. This should however no longer be interesting as neither AdaGrad nor fixed learning rates are feasible methods in online nonstationary settings. AdaGrad has no way of increasing the learning rate, making it adapt poorly to nonstationary data. And fixed learning rates can provide good results in parts of data streams, but do not adapt to the incoming data, and also require cross validation or other techniques to be initialised properly. Therefore the identified strengths and weaknesses of PSGD will be the focus of the following discussion along with possible improvements.

In Figures 45-47 the learning rates of experiments M1-M3 are plotted separately. The signal to noise ratio, R, is also plotted for each example as a reference. The learning rate plots for experiments M1 and M3 indicate that the proposed PSGD implementation is more adaptive towards increases in the variance of the observed responses, than for variance decreases (exp. M2). This is an adverse consequence of the suggested

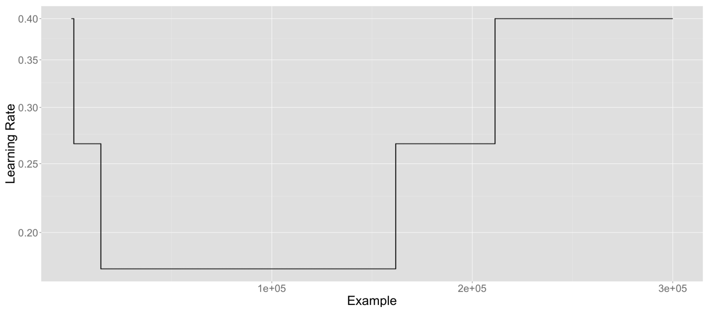
[42]See appendix C.2 for cumulative error plots confirming this.

sequential test for the learner with the smallest PV error in equations (6.4) and (6.5). The tracked mean one-step ahead prediction error of the processes only see relatively small changes if the variance of the observations is reduced. This is essentially because the scaling of the observations has been reduced, and therefore also the size of the errors. However, when there are smaller fluctuations in the PV errors of the processes the standard deviation of the processes is reduced, thereby reducing the acceptance region (keeping the current learner). Though this is true[43], this effect is lagged since many of the observations that are sequentially included to evaluate the estimates of the PV error means and standard deviations, came from a regime with higher variance. This reduces the adaptability of the mean and standard deviation estimates, and therefore also the ability of PSGD to quickly switch learner.

[43]This is the reason why the discussed implementation of PSGD eventualy switches process.

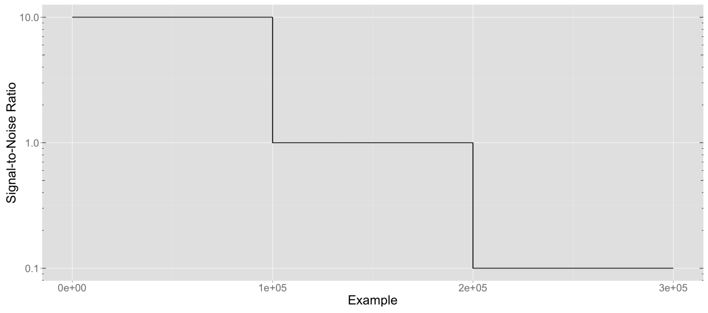


(a) Signal to noise ratio, R, for exp. M1. The steps are $0.1, 1, 10$.

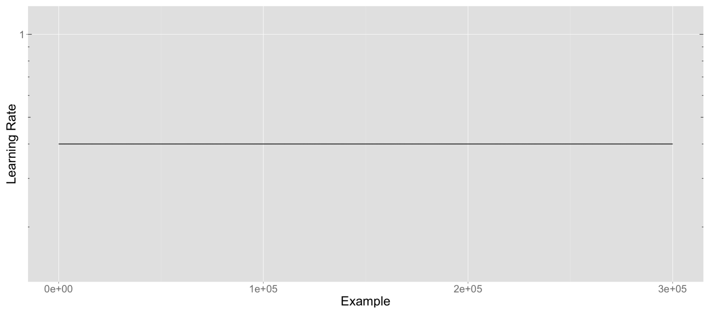


(b) PSGD learning rates for exp. M1.

Figure 45: PSGD learning rates, α, and signal to noise ratios, R, for exp. M1 initialised at $R = 0.1$ and $\alpha = 0.4$. PSGD quickly detects that the learning rate should be reduced in the first signal-to-noise-regime. As R is increased in the second third of the examples, and thereby also the variance of the observations, PSGD requires to process more examples than before to scale up the learning rate. This is because there is a new optimal learner only after the regime change point and the tracked mean of the progressive validation errors of the upper learner needs to compensate for not being the smallest one before the change in variance. This is illustrated in Fig. 48b. The same effect is present for the last increase of R, but is less pronounced since there are fewer examples in the middle regime since the last change, than was the case for the first regime.

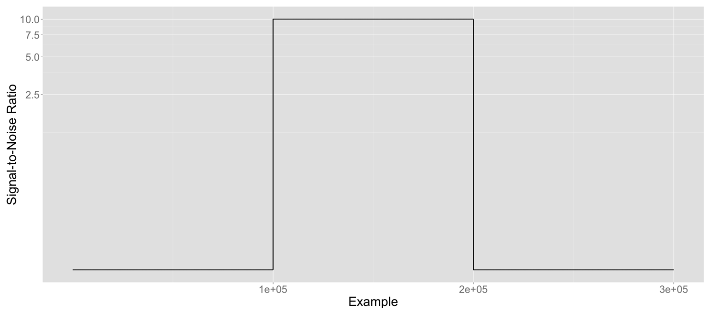


(a) Signal to noise ratio, R, for exp. M2.

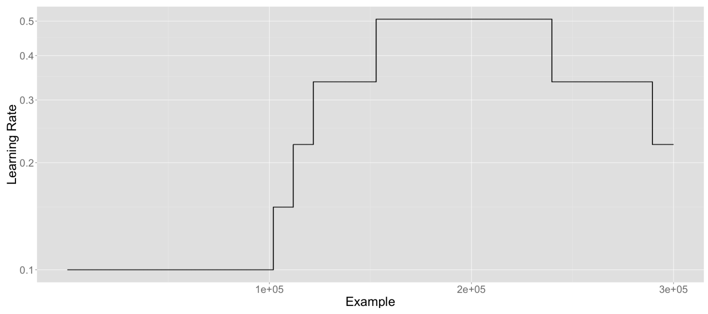


(b) PSGD learning rates initialised at $\alpha = 0.4$, for exp. M2.

Figure 46: PSGD learning rates, α, and signal to noise ratios, R, for exp. M2 initialised at $R = 10$ and $\alpha = 0.4$. From Figure 45 it transpires that Exp. M1 also ends up with learning rates of $\alpha = 0.4$ for $R = 10$, but that the lower values of R should in fact prompt reduced learning rates. Despite the decreases in the variance of the observations, the learning rate remains constant over the $n = 3 \cdot 10^5$ examples. This performance is suboptimal as it leads to overfitting. The reason for the behaviour is that the examples for the first period with higher variance produces errors that are greater in magnitude than the errors from the examples simulated with a smaller variance. As a result the tracked mean is biased by these larger errors and less adaptive to the incoming data with smaller variance. Of course, the fact that the incoming observations have smaller variances gradually reduces the tracked standard deviation of the processes, thereby reducing the acceptance region of the current learner and prompting a switch of the learner. But this effect is also delayed by the larger standard deviation estimates from the initial regimes.



(a) Signal to noise ratio, R, for exp. M3.

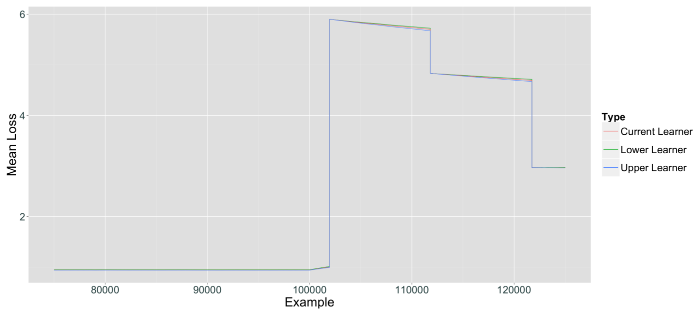


(b) PSGD learning rates for exp. M3.

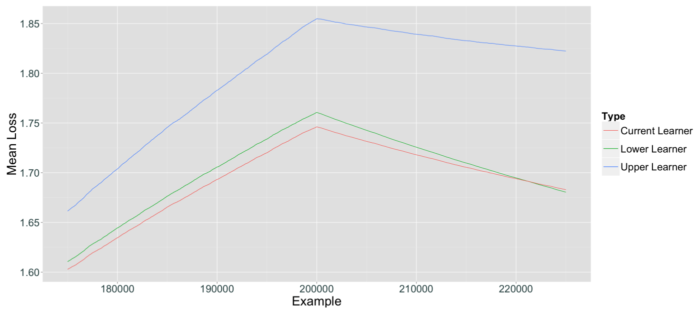
Figure 47: PSGD learning rates, α, and signal to noise ratios, R, for exp. M3 initialised at $R = 0.1$ and $\alpha = 0.1$. The learning rate initialisation is clearly suitable for the first third of the data set, as there are no changes to the learning rate. As the signal to noise ratio, R, is increased, and thereby also the variance of the observations in the second third of the examples, the learning rate quickly adapts to the new regime. This behaviour is very promising as this kind of learning rate adaptiveness prevents underfitting the model to the incoming data. The adaptiveness is due to the larger errors caused by the variance increase. As the signal to noise ratio is reduced again there is a considerable lag before the learning rates see gradual reductions. This is because a variance reduction reduces the size and fluctuations of the ensuing errors, meaning they have a smaller impact on the sequentially updated mean and standard deviation of the errors. This phenomenon is also discussed in Figure 46.

Another problem is that before and after a regime change different learners might be optimal. If this is the case between two regimes, then the mean PV error of the ideal learner in the first regime will shrink more compared to the other learners, until the regime change. After this point, a different learner is optimal, but needs to overtake the advantage gained by the other candidate process(es) in the previous regime. This phenomenon is illustrated in Figure 48. Bear in mind that this does not prevent PSGD adaptability. It rather implies that the PSGD learning rates adapt slower after a long streak of data coming from very similar generative models that is followed by an abrupt change in the variance of the observations.

These adaptability problems are addressed in the next section.



(a) Progressive validation errors of the three learners around the first change point for exp. M3.



(b) Progressive validation errors of the three learners around the second change point for exp. M3.

Figure 48: The sequentially updated progressive validation errors for the examples around the two change points seeing the signal to noise ratio, R, respectively increased and decreased. Before the second change point in Fig. 48b, the current learner has the lowest mean error, a trend that is reinforced towards the change point. After the change point, R is reduced, thereby reducing the variance of the observations as well as the sizes of the errors. This is evidenced by the slower rate of change in the sequentially updated mean errors. After the second change point the lower learner reduces its mean error the fastest, but is required to process a lot of examples before overtaking the current learner and be switched to the current learner. This problem is less important when R is scaled up at the first change point as then the increased variance of the observations conduce to higher errors which have a greater impact on the mean of the learners after the change point. The current adaptation of PSGD therefore see learning rates adapt more quickly when facing increases in R.

6.5 Conclusion and Further Work

Conclusion. The experiments on the proof-of-concept implementation of PSGD indicate that PSGD is clearly superior to AdaGrad and SGD with a fixed learning rate as it can adapt to nonstationary data on the fly without requiring cross validation for learning rate initialisation. Additionally the adaptiveness to eventual changes in the generative model of the incoming data prevents the model from degrading over time.

The increased memory constraints of PSGD are low as the estimates of the upper and lower learners are only sequentially stored. In order to perform the sequential tests deciding whether or not to switch learner, one only needs to store the mean and standard deviation of the one step ahead prediction error of the three learners. This is a small price to pay for the increased adaptiveness as the RAM of most computers will be able to handle this in most cases.

The method in the proposed framework is computationally feasible as it only computes the gradient of the loss once per point, regardless of how many learners it is running in parallel. The method is parallelisable and the added computational steps are not notably time consuming. The slightly increased run-time of PSGD, compared to SGD, should not outweigh the benefits of its adaptiveness to nonstationary data.

The experiments of the PSGD implementation on the nonstationary data that was sampled from different generative models over time in section 6.4.3 showed promising results of adaptive learning rates, but also permitted the identification of weaknesses. It is shown and explained that PSGD, as implemented here, can struggle when the variance of the observed responses decrease. Additionally, it is shown and explained that after a long streak with the same learning rate, PSGD can be slow to adapt its learning rates to the new regime.

Due to time constraints these weaknesses were not addressed directly, but potential solutions have been suggested. The overall conclusion is therefore that conceptually the PSGD framework shows great promise for online learning on nonstationary data due to the good performance indications of the proof-of-concept implementation, its relative simplicity and computational scalability. Methods in this framework can therefore be the focus of further research and potentially be included in online learning softwares such as Vowpal Wabbit.

Further work. A sequential test[44] for detecting variance change points in the stream of observed responses could be designed and implemented. It should subsequently be verified whether resetting the PV errors of the learners at identified change points will increase the adaptiveness of the method sufficiently to warrant an extension of the proof-of-concept implementation. This suggestion seems to address the two main problems identified. Firstly, it means that after a regime change, the difference of the scales of PV errors between regimes no longer matters since the tracked PV errors are reset. And secondly, since the tracking means are reset, there is no longer a problem that one of the processes has an accumulated error advantage from a previous regime with a different response variance. Due to time constraints a specific sequential variance test for the observed responses was not designed nor implemented. Initial investigations suggest that the monograph by Csörgö and Horváth (1997) may include relevant sequential tests for this application.

If it turns out that sequential detection of variance change points does not provide the desired results, since it could fail to detect broad enough nonstationarity behaviours, then other remedies can be considered. The core problem identified in section 6.4.3, is that in cases where the current learner is optimal for many data points, the adaptiveness decreases since very many examples have been used to compute the mean and standard deviation of the PV errors. In other words, the proposed algorithm lacks an adaptive mechanism for forgetting past observations. Intuitively, one could suggest exponential decay of the weighting of the contributions of

[44]As discussed previously, sequential tests are generally computationally efficient, a key constraint in this problem setting.

past PV errors, but this does not seem scalable due to memory constraints. A more scalable approach could therefore be to store a window of the last n_w PV errors. This approach could be used to either compute exponential decay on the oldest errors, or as an unaltered error vector from which the means and standard deviations can be computed. The n_w parameter will clearly influence the performance of the method, motivating further research on its effect.

The proposed PSGD framework is not restricted to running three learners in parallell. More work should be done to confirm whether three is indeed the optimal number of learners, and how the scaling factors should be chosen.

It should be verified that the promising results presented in this section still hold for a large scale implementation of PSGD that handles more parameters in the training model.

The switching rules presented in equations (6.4) and (6.5) are approximative results that are shown to perform well. One should nevertheless examine if a more precise result can be derived from realistic assumptions, or if other switching rules with better performance can be developed.

Furthermore one should derive and implement sequential approximate confidence intervals of the point estimates. These should be able to inform a PSGD user of more uncertain estimates, possibly due to periods of changes in the generative models.

One should also formalize properties of this method under realistic assumptions. This especially includes deriving regret bounds and convergence properties for reasonable nonstationary settings.

Additionally, a more precise comparison of the running time of PSGD compared to that of the SGD variants implemented in `vw` could be interesting. Note however that, as previously explained, the result of this should not be a key factor in determining the preferability of PSGD over existing SGD variants.

7 Summary

This thesis has studied online learning using SGD-based methods implemented in the statistical software Vowpal Wabbit. As such, selected methods and concepts that motivate, explain and build on the SGD algorithm have been treated in sections 2 and 3. These methods, and the theoretical results that motivate them, are derived for stationary settings where the generative models of incoming observations are not changing over time.

One of these methods, AdaGrad, an extension of SGD and selected by default in `vw`, was the focus of experiments in static settings. These experiments were detailed in section 4. The results in section 5.1 confirmed the excellent performance of AdaGrad in stationary settings, while indicating that the high noise levels can weaken the performance. Comparing the performance of AdaGrad to `R`'s `lm` function, a batch learning method, shows that the price of AdaGrad's computational efficiency is less precise estimates.

One of the main advantages of online learning methods is that they scale very well in scenarios where extremely large amounts of data is continuously arriving, e.g. user data for a popular website or mobile application. The usefulness of online learning is therefore greatly limited if the methods perform poorly on nonstationary data. AdaGrad is intended for stationary settings by design as its learning rates are continuously reduced, thus reducing its adaptability to nonstationarities in models generating the incoming data. Currently one is therefore required to preselect a fixed learning rate if one wishes to use `vw` on stationary data. This is suboptimal since the performance of SGD with fixed learning rates depends on the variance of the incoming data. As a result, even after hypothetically selecting an optimal learning rate by cross validation, one may end up overfitting or underfitting the model to the data, on parts of the data. Furthermore, cross validation is a batch learning technique that is not designed to be used sequentially implying that the solution would not scale well.

In section 5.2 we clearly demonstrate these problems. Namely that AdaGrad is unsuitable for model fitting in nonstationary settings, and that SGD with fixed learning rates comes at the cost of lacking adaptibility in nonstationary settings.

As a response to these problems, and given the constraints of requiring a highly scalable method, we propose a new framework that builds on the SGD algorithm in section 6. Our framework, Parallelised SGD, or PSGD, consists of running two or more alternative SGD-learners in parallell to a chosen SGD learner for every example. The alternative learners have learning rates scaled from the value of the learning rate of the chosen learner. This provides a scalable framework as the gradient still only needs to be computed once per example, and the added cost of the alternative learners can be diminished through efficient parallelisation.

The alternative learners can be used to tune the learning rates by sequentially comparing their errors. To preserve the scalability of the method we propose to sequentially update the mean and standard deviations of the one-step ahead prediction errors, and use this to switch learner when an alternative learner has a statistically significant smaller error than the current learner. Experiments on a proof-of-concept implementation demonstrates that PSGD is superior to AdaGrad and SGD with fixed learning rates in nonstationary settings. However, further work is required to improve the adaptiveness of the method to a wider range of nonstationarities. A discussion of the identified weaknesses of the proof-of-concept implementation and possible solutions can be found in section 6.5.

Continued research on this framework shows great potential for yielding a class of adaptive learners that can automatically handle nonstationary data and be subject to large scale implementations in online learning software such as Vowpal Wabbit.

References

[1] R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] N.H. Bingham and J.M. Fry. *Regression: Linear Models in Statistics*. Springer, 2010.

[3] A. Blum, A. Kalai, and J. Langford. "Beating the Hold-Out: Bounds for K-fold and Progressive Cross-Validation". In: *Proceedings of the twelfth annual conference on Computational learning theory* (1999).

[4] L. Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of Computational Statistics* (2010), pp. 177–186.

[5] G. Casella and R. Berger. *Statistical Inference*. Second edition. Duxbury, 2001.

[6] A. Cauchy. "Méthode générale pour la résolution des systèmes d'équations simultanées". In: *C.R. Acad. Sci. Par* **25** (1847), pp. 536–538.

[7] Kim Chu C. et al. "Map-reduce for machine learning on multicore". In: *Advances in neural information processing systems* **19** (2007), pp. 281–288.

[8] Cisco. "The Zettabyte Era: Trends and Analysis". In: *Cisco Visual Networking Index (VNI)* (1965).

[9] M. Csörgö and L. Horváth. *Limit Theorems in Change-Point Analysis*. Wiley Series in Probability and Statistics, 1997.

[10] P. Domingos. "A Unified Bias-Variance Decomposition and its Applications". In: *7th proceedings of the International Conference on Machine Learning* (2000), pp. 231–238.

[11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *The Journal of Machine Learning Research* **12** (2011), pp. 2121–2159.

[12] E. Fox. "Case Study: Estimating Click Probabilities". In: *Lecture notes of STAT548 - Machine learning for Big Data* (2014).

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[14] W. Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* **58** (Mar. 1963), pp. 13–30.

[15] N. Karampatziakis and J. Langford. "Online Importance Weight Aware Updates". In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence* (2011), pp. 392–399.

[16] D.E. Knuth. *The Art of Computer Programming, vol. 2: Seminumerical Algorithms*. 3rd. Boston: Addison-Wesley, 1998.

[17] J. Langford, L. Li, and T. Zhang. "Sparse Online Learning via Truncated Gradient". In: *J. Mach. Learn. Res.* **10** (Mar. 2009), pp. 777–801.

[18] G.E. Moore. "Cramming More Components onto Integrated Circuits". In: *Electronics Magazine* (1965).

[19] N. Murata. "A Statistical Study on On-line Learning". In: *On-line learning in neural networks* (1998), pp. 63 –92.

[20] J. Nocedal and S.J. Wright. *Numerical Optimization*. Second edition. Springer, 2006.

[21] H. Robbins and S. Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* **22**.3 (1951), pp. 400–407.

[22] S. Ross, P. Mineiro, and J. Langford. "Normalized online learning". In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence* (2013).

[23] G. Salton and C. Buckley. "Term weighting approaches in automatic text retrieval". In: *Information Processing and Management* 24 (1988).

[24] C. Snijders, U. Matzat, and U.-D. Reips. "'Big Data': Big gaps of knowledge in the field of Internet". In: *International Journal of Internet Science* **7** (2012), pp. 1–5.

[25] N. Vaits, E. Moroshko, and K. Crammer. "Second-Order Non-Stationary Online Learning for Regression". In: *Journal of Machine Learning Research* 16 (2015), pp. 1481–1517.

[26] M. Zinkevich. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent". In: *Proceedings of the Twentieth International Conference on Machine Learning* (2003), pp. 928–936.

[27] H. Zou and T. Hastie. "Regularization and variable selection via the Elastic Net". In: *Journal of the Royal Statistical Society, Series B* **67** (2005), pp. 301–320.

Appendices

A Implementation of Experiments

The workflow consists of calling `Python` files and `R`-functions from a master project in `R`.

Simulate data and save R objects in csv files. Data is simulated as explained in sections 4.1.1-4.1.2. The `R` objects are saved in csv text files locally using the `write.csv` function in `R`.

Convert csv files to `vw` file format. In order to run `vw` on the data, the data files must be in the Vowpal Wabbit file format[45]. Reformatting scripts were written in `Python`.

Run `vw`. Vowpal Wabbit can now be run by invoking terminal commands with the `system` function in `R`.

Convert `vw` output to csv files. `Python` scripts were written to carry out this task.

Read csv files into `R`. The `read.csv` function can now be used to import the `vw` output into `R` objects.

Post-experimental data analysis. The experimental results stored in `R` objects are subsequently used for data analysis, primarily by producing the figures of this thesis.

[45]See `https://github.com/JohnLangford/vowpal_wabbit/wiki/Input-format`

B Derivations for Linear Model Experiments

We have that

$$\widehat{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon} \tag{B.1}$$

where the following random variables are independent

$$\epsilon_j \sim N(0, \sigma_\epsilon^2) \quad \beta_j \sim N(\mu_\beta, \sigma_\beta^2) \quad X_{ij} \sim U(x_{\min}, x_{\max}) \forall i, j$$

Let μ_x and v_x denote the mean and variance of a uniform random variable respectively, as follows

$$\begin{aligned} \mu_x &= \mathrm{E}(X_{ij}) = \frac{x_{\max} + x_{\min}}{2} \\ v_x &= \mathrm{Var}(X_{ij}) = \frac{(x_{\max} - x_{\min})^2}{12} \end{aligned} \tag{B.2}$$

Moments and Signal to Noise Ratio

Expected values follow from the above

$$\begin{aligned} \mathrm{E}[\widehat{y}_i] &= \sum_{j=1}^{p} \mathrm{E}[X_{ij}\beta_j] + \mathrm{E}[\epsilon_i] \\ &= p\big(\mathrm{E}[X_{ij}]\,\mathrm{E}[\beta_j]\big) + 0 \\ &= p\mu_x\mu_\beta \qquad \text{(B.3)} \\ &= 0 \quad \text{for } \mu_\beta = 0 \qquad \text{(B.4)} \end{aligned}$$

And the variance of the responses are computed as follows $\forall i \in \{1, ..., n\}$

$$\begin{aligned} \mathrm{Var}[\widehat{y}_i] &= \sum_{j=1}^{p} \mathrm{Var}[X_{ij}\beta_j] + \mathrm{Var}[\epsilon_i] \\ &= p(v_x\sigma_\beta^2 + v_x\mu_\beta^2 + \mu_x^2\sigma_\beta^2) + \sigma_\epsilon^2 \\ &= p\big(\sigma_\beta^2(\mu_x^2 + v_x) + v_x\mu_\beta^2\big) + \sigma_\epsilon^2 \\ &= p\sigma_\beta^2 + \sigma_\epsilon^2 \quad \text{for } \mu_\beta = 0 \text{ and } v_x + \mu_x^2 = 1 \qquad \text{(B.5)} \end{aligned}$$

By using the condition in equation (B.5) one can derive a simple expression for R, defined in equation (4.9) as $R = \sigma_{\text{noise}}/\sigma_{\text{signal}}$. Note that $\sigma^2_{\text{noise}} = \sigma^2_\epsilon$ and $\sigma^2_{\text{signal}} = \mathrm{Var}(\sum_{j=1}^{p} X_{ij}\beta_j) = p\sigma^2_\beta$, for any i, as transpires from equation (B.5). The expression for R is now simply

$$R = \frac{\sigma_\epsilon}{\sqrt{p}\sigma_\beta} \quad \text{for } \mu_\beta = 0 \text{ and } v_x + \mu_x^2 = 1 \tag{B.6}$$

One can solve $v_x + \mu_x^2 = 1$ for the parameters of the Uniform distribution $(x_{\min}, x_{\max})$ by for example setting $\mu_x = 1/2$ and $v_x = 3/4$. This yields

$$(x_{\min}, x_{\max}) = (-1, 2) \tag{B.7}$$

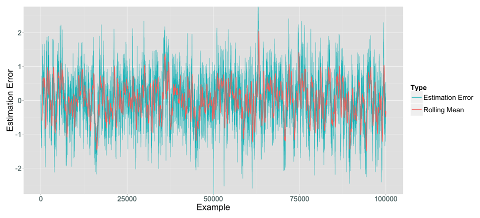
Note that this implies that equation (B.5) reduces to

$$\mathrm{Var}[\widehat{y}_i] = \sigma^2_\epsilon \left(1 + \frac{1}{R^2}\right) \quad \text{for } \mu_\beta = 0 \text{ and } v_x + \mu_x^2 = 1 \tag{B.8}$$

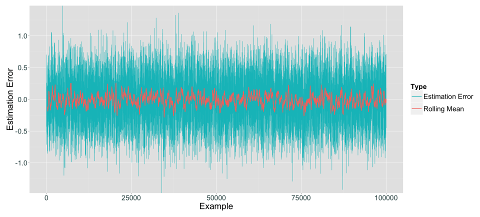
The Central Limit Theorem [5] can now be applied to derive confidence intervals for the means of the errors, but as this is not essential for the message of the thesis it is omitted.

C Additional Figures

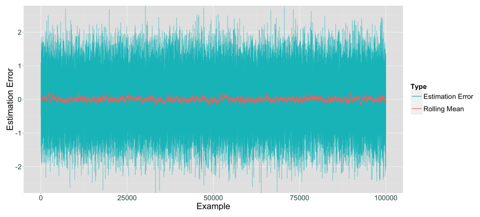
C.1 Error Plots for Exp. D4-D6



(a) Exp. D4: Estimation errors with rolling mean curves. $\alpha = 0.005$.

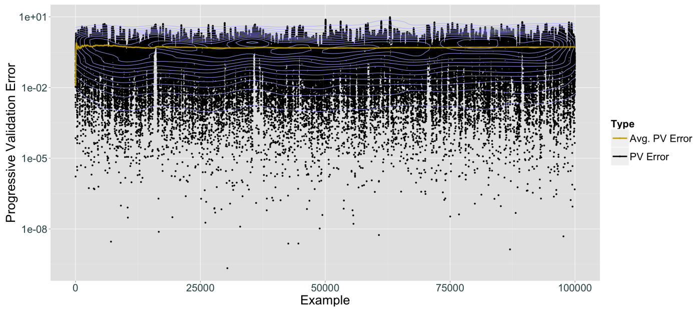


(b) Exp. D5: Estimation errors with rolling mean curves. $\alpha = 0.05$.

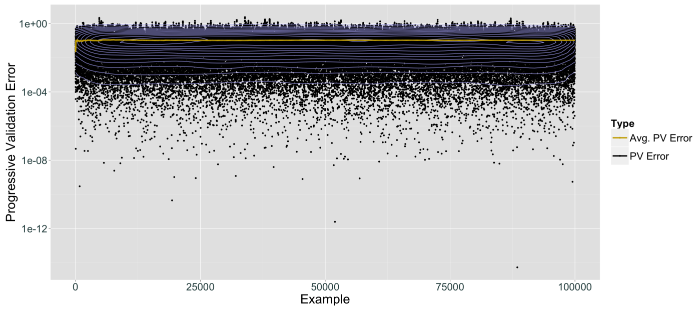


(c) Exp. D6: Estimation errors with rolling mean curves. $\alpha = 0.5$.

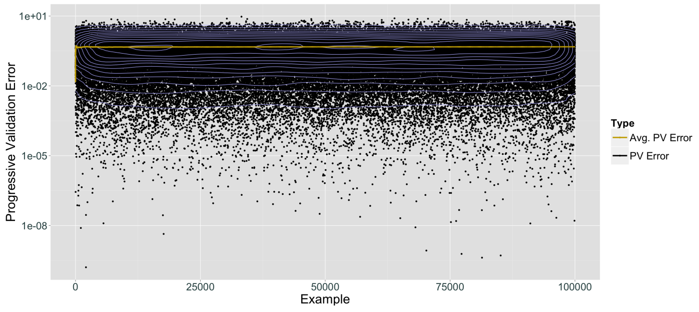
Figure 49: Estimation errors for the tested values of the learining rate, α. There are considerable fluctuations in the errors of model Exp. D4. This supports the fact that the model is underfitting the model to the data due to a too small learning rate. On the other extreme there is Exp. D6 with seemingly stationary errors. This indicates model overfitting and too elevated α. The errors of Exp. D5 show more promising behaviour.



(a) Exp. D4: Pointwise and mean progressive validation errors. $\alpha = 0.005$.

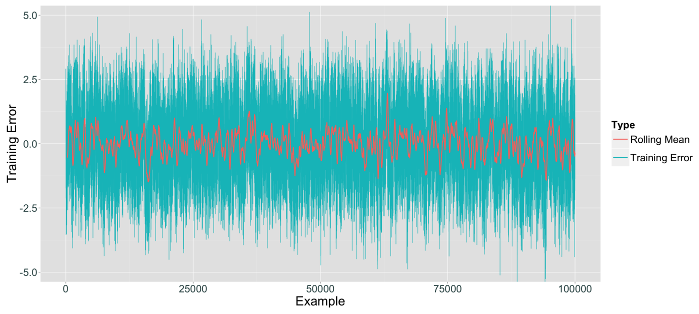


(b) Exp. D5: Pointwise and mean progressive validation errors. $\alpha = 0.05$.

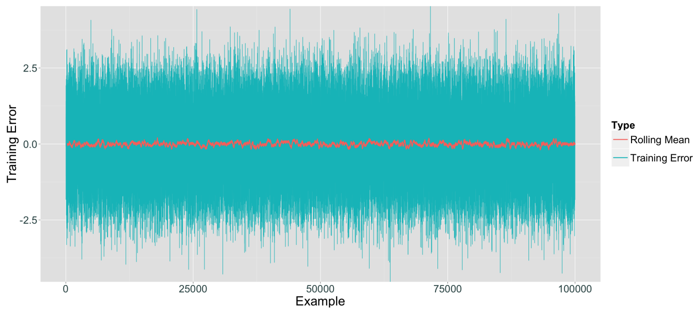


(c) Exp. D6: Pointwise and mean progressive validation errors. $\alpha = 0.5$.

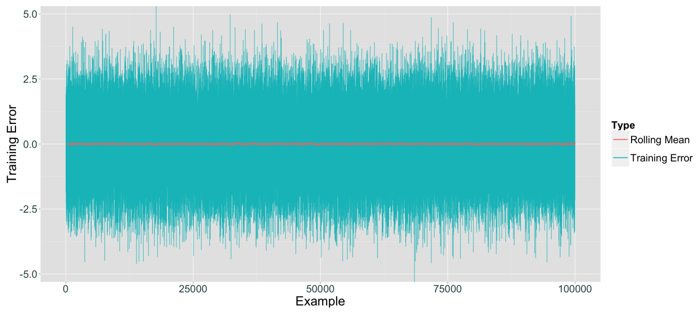
Figure 50: Pointwise and average progressive validation errors for the tested values of the learining rate, α. Due to poor axis scaling it is hard to tell whether there are differences between the error trends of the experiments. However, as expected, the PV errors of Exp. D2 seem to be smallest on the whole. This indicates that it is the least affected by model overfitting or underfitting.



(a) Exp. D4: Training errors with a rolling mean curve.

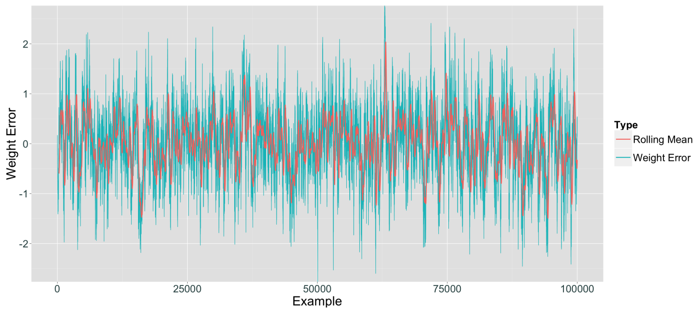


(b) Exp. D5: Training errors with a rolling mean curve.

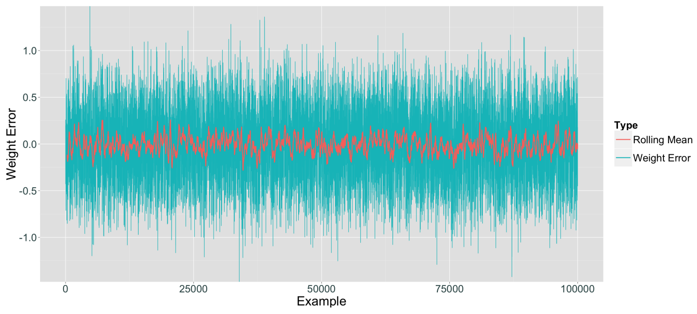


(c) Exp. D6: Training errors with a rolling mean curve.

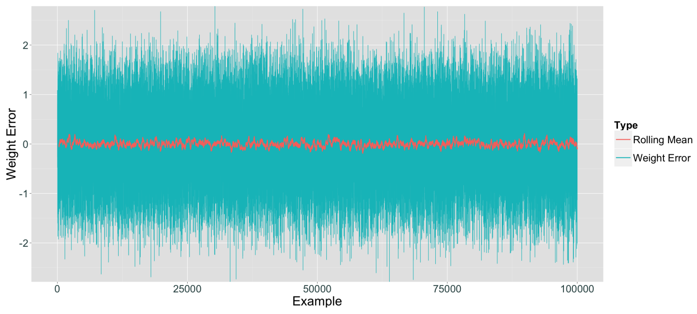
Figure 51: Training errors for the tested values of the leariŋing rate, α. There are considerable fluctuations in the errors of model Exp. D4. This supports the fact that the model is underfitting the model to the data due to a too small learning rate. On the other extreme there is Exp. D6 with seemingly stationary errors. This indicates model overfitting and too elevated α. The errors of Exp. D5 show more promising behaviour.



(a) Exp. D4: Weight estimation errors.



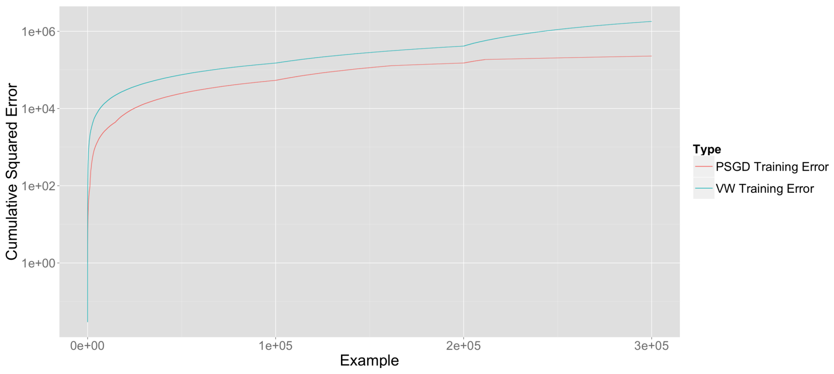
(b) Exp. D5: Weight estimation errors.



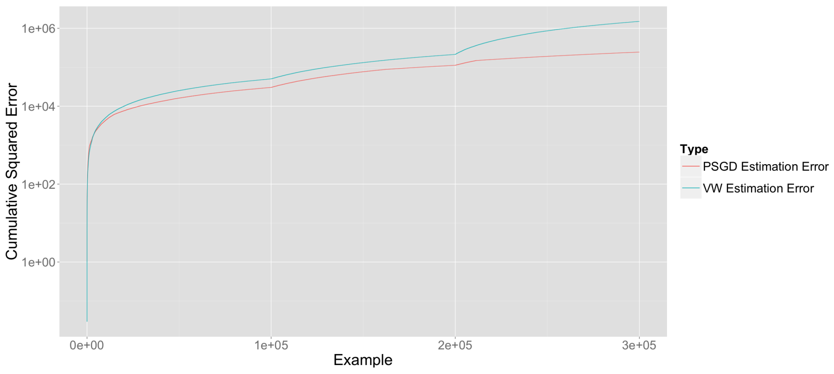
(c) Exp. D6: Weight estimation errors.

Figure 52: Weight estimation errors for the tested values of the learining rate, α. There are considerable fluctuations in the errors of model Exp. D4. This supports the fact that the model is underfitting the model to the data due to a too small learning rate. On the other extreme there is Exp. D6 with seemingly stationary errors. This indicates model overfitting and too elevated α. The errors of Exp. D5 show more promising behaviour.

C.2 Error Plots for Exp. M1-M3

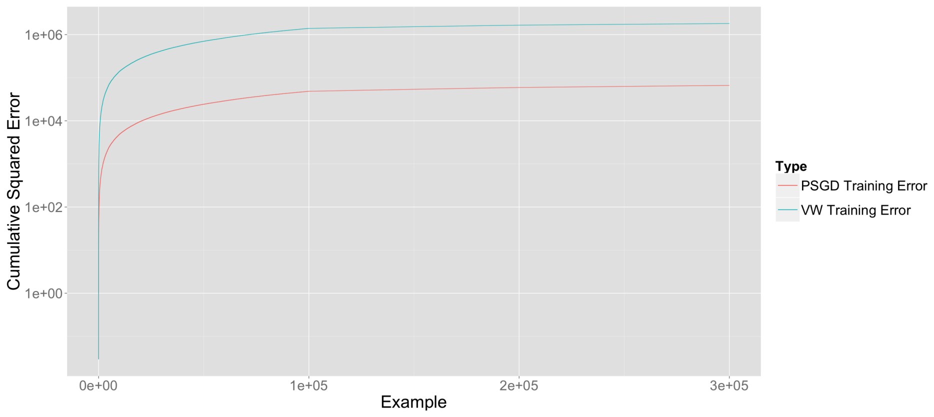


(a) Cumulative training errors of PSGD and `vw` with fixed learning rates.

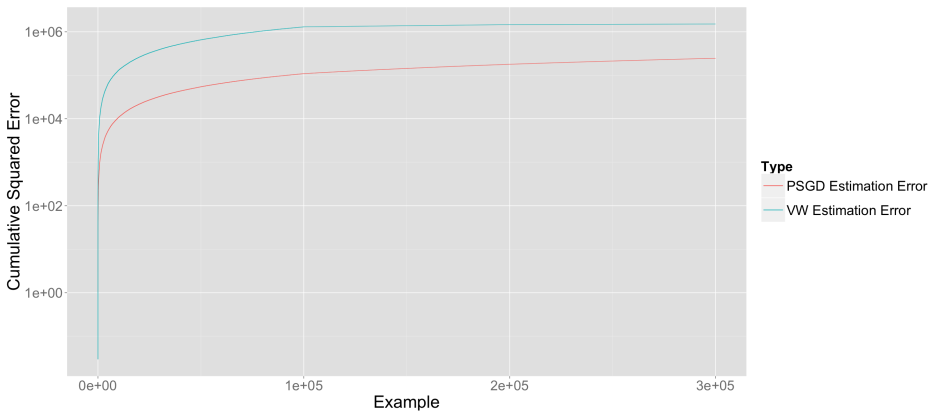


(b) Cumulative estimation errors of PSGD and `vw` with fixed learning rates.

Figure 53: Exp. M1: Cumulative estimation and training errors of the PSGD method are superior to those of `vw` with a fixed learning rate, due to the learning rate adjustments in nonstationary settings. This indicates that the PSGD method manages to adapt to the data on the fly unlike SGD with fixed learning rates. Note that `vw` and PSGD have the same initial learning rate, $\alpha_i = 0.4$.

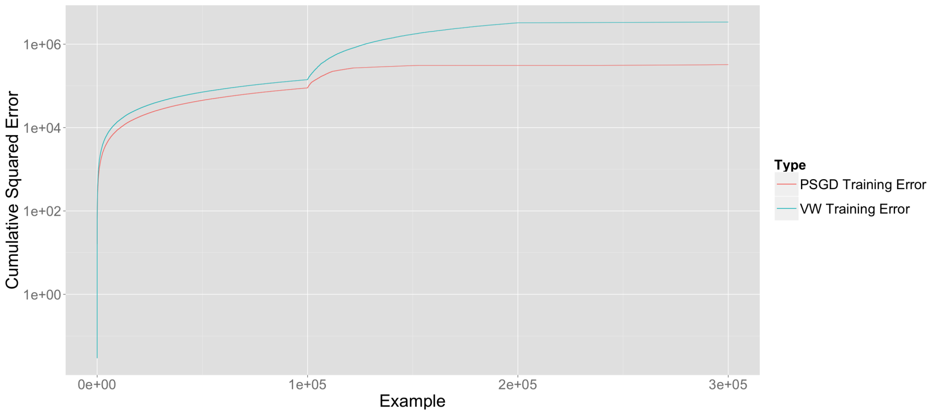


(a) Cumulative training errors of PSGD and vw with fixed learning rates.

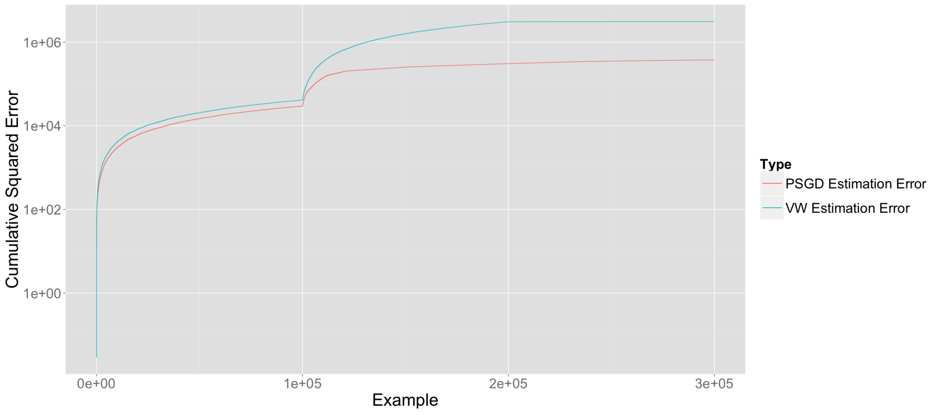


(b) Cumulative training errors of PSGD and vw with fixed learning rates.

Figure 54: Exp. M2: Cumulative estimation and training errors of the PSGD method are superior to those of vw with a fixed learning rate, due to the learning rate adjustments in nonstationary settings. This indicates that the PSGD method manages to adapt to the data on the fly unlike SGD with fixed learning rates. Note that vw and PSGD have the same initial learning rate, $\alpha_i = 0.4$.



(a) Cumulative training errors of the PSGD and AdaGrad method.



(b) Cumulative estimation errors of the PSGD and AdaGrad method.

Figure 55: Exp. M3: Cumulative estimation and training errors of the PSGD method are superior to those of `vw` with a fixed learning rate, due to the learning rate adjustments in nonstationary settings. This indicates that the PSGD method manages to adapt to the data on the fly unlike SGD with fixed learning rates. Note that `vw` and PSGD have the same initial learning rate, $\alpha_i = 0.1$.