# Second Order Techniques for Learning Time-series With Structural Breaks

**Takayuki Osogami**
IBM Research - Tokyo
osogami@jp.ibm.com

## Abstract

We study fundamental problems in learning nonstationary time-series: how to effectively regularize time-series models and how to adaptively tune forgetting rates. The effectiveness of L2 regularization depends on the choice of coordinates, and the variables need to be appropriately normalized. In nonstationary environment, however, what is appropriate can vary over time. Proposed regularization is invariant to the invertible linear transformation of coordinates, eliminating the necessity of normalization. We also propose an ensemble learning approach to adaptively tuning the forgetting rate and regularization-coefficient. We train multiple models with varying hyperparameters and evaluate their performance by the use of multiple hyper forgetting rates. At each step, we choose the best performing model on the basis of the best performing hyper forgetting rate. The effectiveness of the proposed approaches is demonstrated with real time-series.

## Introduction

Time-series data is ubiquitous in industries, where the quality and efficiency of services rely on learning from such data for forecasting, anomaly detection, and decision making. For example, financial institutions are eager to make better forecast in stock markets. Early detection of anomaly from sensor data is critical in complex industrial products.

Learning time-series in industries is often hard due to nonstationarity, or structural breaks in particular. For example, financial markets change drastically with economic bubbles and major exogenous events. Sensor data from industrial products is often susceptible to changes in their environment and deterioration of their components. Thus, there has been a significant amount of research to deal with structural breaks in time-series in the field of artificial intelligence, statistics, signal processing (Farhang-Boroujeny 2013), and economics (Granger and Newbold 1986).

A common idea in the literature is not only to incorporate latest data but also to gradually forget past data. The most celebrated is the method of recursive least squares, which recursively updates the weights (coefficients) of a linear model in a way that the weighted squared error of prediction is minimized, where the weight on the past data gets smaller exponentially with time (Farhang-Boroujeny 2013). Stochastic gradient methods also follow this idea when they are applied to online learning for time-series.

Although forgetting the past is essential in dealing with structural breaks, it involves two difficulties in practice. First, one needs to set (and adaptively tune) the forgetting rate appropriately. If we do not forget sufficiently, we cannot adapt to changes. If we forget too much, we might overreact to noise and trivial changes. Second, the size of effective data is small when we forget the past. Without appropriate regularization, our models can overfit to the small data.

We propose a technique for adaptively changing the forgetting rate and the strength of regularization to deal with structural breaks. We simultaneously train a small number of models with varying forgetting rates and regularization-coefficients. We then track the predictive errors of those models with multiple values of the hyper forgetting rate, placing higher weight on more recent predictive errors. Our key idea is to first choose the best performing hyper forgetting rate and then choose the best performing model based on the best hyper forgetting rate at each (time) step. The best model is then used to make a prediction.

Notice that, in the standard literature, the forgetting rate is a hyperparameter *of an objective function* and cannot be tuned in a way similar to the hyperparameters of machine learning models and algorithms. We need a criteria to determine what forgetting rates are good, but the forgetting rate is a part of the objective function. We introduce hyper forgetting rate as a means to determine what forgetting rates are good. With a hyper forgetting rate, the forgetting rate can now be considered as a hyperparameter of a model.

Our approach is primarily targeted at industrial time-series that can exhibit structural breaks but have small to medium dimensions (after feature engineering and variable selection). For such time-series, a compact linear model having at most $n = 1,000$ weights often suffices or even works best, as complex models either overfit to latest data or fail to adapt to changes. We thus design our approach by allowing $O(n^2)$ computational time (and space) in updating our models and making a prediction at each (time) step. However, we keep the computational time per step independent of the length of time-series, which may be indefinitely long.

The $O(n^2)$ time per step is sufficient to optimize the weights for a linear model with a given forgetting rate and no regularization (by recursive least squares). However, this

optimization becomes difficult with regularization, as discussed in Tsakiris (2010); Tsakiris, Lopes, and Nascimento (2010). In addition, the effectiveness of standard regularization such as L2 is sensitive to transformation of coordinates. Namely, how much L2 regularization reduces the absolute value of each parameter depends on how the data is normalized. If the magnitude of a variable varies over time, L2 regularization would not have the expected effect. While we might be able to preprocess the data with variable selection and normalization (*e.g.*, letting mean be 0 and variance be 1), the scale of the data might change over time. One could renormalize the data if the past data is stored, but then leaning cannot be performed in $O(n^2)$ time per step.

We thus also propose a variant of L2 regularization where the L2 norm is defined with the Hessian of the weighted squared error, which is the loss function without regularization. The proposed regularization keeps the learning problem invariant to invertible linear transformations (namely, the optimal weights are contra-variate to invertible linear transformations). Also, the optimal weights with the proposed regularization can be updated in $O(n^2)$ time per step.

Our primary contributions are two techniques: (i) the ensemble learning method with following the best hyper forgetting rate for adaptively tuning hyperparameters of a nonstationary time-series model and (ii) the regularization that is invariant to invertible linear transformations. We empirically demonstrate the effectiveness of the proposed techniques with real time-series datasets. In addition, we propose a technique of rank-one update for the pseudo-inverse (Moore-Penrose generalized matrix inverse) of a symmetric matrix, which we need to use recursively. The proposed approach accumulates numerical error slower than existing methods by orders of magnitude.

## Related Work

Online learning for nonstationary time-series is oftern performed by tuning the learning rate in stochastic gradient descent (SGD). The variance-based SGD (vSGD) is particularly designed for nonstationary data, enabling the learning rates to increase and decrease according to the data (Schaul, Zhang, and LeCun 2013). Cogra is a recent extension of vSGD (Miyaguchi and Kajino 2019). There also exist techniques, including Almeida (Almeida et al. 1999) and Hypergradient descent (Baydin et al. 2017), that seek to learn appropriate values of the learning rates in an online manner. In our experiments, we study how our proposed approach compares against those existing methods as well as against standard SGD methods such as AdaGrad (Duchi, Hazan, and Singer 2011), Adam (Kingma and Ba 2014), and RMSProp (Tieleman and Hinton 2012)

Online learning for nonstationary time-series is also related to regret minimization in online (convex) optimization. Some of the SGD methods such as AdaGrad are known to give optimal regret bounds for some cases (Duchi, Hazan, and Singer 2011; Hazan 2016). While minimizing regret is different from minimizing weighted squared error, they can lead to similar algorithms. Online Newton Step (ONS) (Hazan, Agarwal, and Kale 2007; Agarwal et al. 2006) is related to ours from an algorithmic point of view. A key difference is that ONS uses fixed parameters (specifically, $D, G$, and $\alpha$ in Hazan, Agarwal, and Kale (2007)) that need to be set on the basis of the properties of the objective functions under consideration, while we adaptively change our forgetting rates and regularization-coefficients. MetaGrad trains multiple models with different learning rates and combine their output in a way that regret is minimized (van Erven and Koolen 2016).

Our method of following the best hyper forgetting rate is a Follow The Leader (FTL) method, which has been studied in the context of online learning (Littlestone and Warmuth 1994; Freund and Schapire 1997). While the prior work uses FTL to choose a model from candidates, we use FTL to choose a hyper forgetting rate, which is then used to choose a model, to specifically deal with nonstationarity. By choosing the hyper forgetting rate, we determine the best loss function, which in turn uniquely determines the best model in our approach. At a high level, our approach can thus be seen as the standard setting of online learning, and the theoretical guarantee established for FTL carries over to our approach. One may also replace FTL in our method with more sophisticated techniques such as Follow the Regularized Leader (De Rooij et al. 2014) and Follow the Leading History (Hazan and Seshadhri 2009).

Note that approaches similar to following the best hyper forgetting rate have been studied in the context of choosing the values of hyperparameters (Minku 2019). The novelty in our approach is in tuning hyperparameters by choosing the hyper forgetting rate in a way similar to Minku (2019).

Recursive least squares (RLS) dates back to Gauss (19th century), but the forgetting rate has been set a priori based on domain knowledge, separate data (Van Vaerenbergh, Santamaría, and Lázaro-Gredilla 2012), or theoretical analysis under the assumption of stationarity (Guo, Ljung, and Priouret 1993). With no forgetting, L2 regularization is straightforward with RLS, as it only needs to initialize the Hessian as a diagonal matrix. With forgetting, the effect of initial regularization diminishes over time, and the inverse Hessian can no longer be updated via the Sherman-Morrison lemma (Tsakiris 2010; Tsakiris, Lopes, and Nascimento 2010). Horita et al. (2004) use a rank-one update technique of eigendecomposition, and Gay (1996) regularizes random one dimension at each step. Elisei-Iliescu et al. (2017) uses a dichotomous coordinate descent method. These techniques can be performed in $O(n^2)$ time per step, but their effectiveness is sensitive to the selection of coordinates.

Garrigues and Ghaoui (2009) study an online algorithm for Lasso, where a homotopy algorithm is used to update the optimal solution to Lasso when new data is given. Although their algorithm runs efficiently once a relevant subset of data is extracted, it stores the entire data that has been observed. The computational complexity thus grows linearly with the length of data, and it is not suitable for industrial time-series that continue indefinitely. They also study a way to adaptively tune the regularization-coefficient, which however cannot be directly applied to our case in $O(n^2)$ time, as it would require recomputing relevant matrices when the forgetting rate is changed. Lafond, Wai, and Moulines (2016) study another online algorithm for Lasso

but consider a stationary setting where an independent and identically distributed (i.i.d.) pair of a response and corresponding explanatory variables is observed one at a time in a sequential manner. In particular, they use a monotonically decreasing learning rate, which is unsuitable for nonstationary time-series.

## Regularized Recursive Least Squares

We study the problem of learning a nonstationary time-series model in an online manner. Let $f_{\boldsymbol{\theta}_t}(\cdot)$ be the model, which is nonstationary in the sense that it has time-varying weights, $\boldsymbol{\theta}_t$ at step $t$. Given an input $\mathbf{x}_t$ at $t$, the model makes a prediction $\hat{y}_t \equiv f_{\boldsymbol{\theta}_t}(\mathbf{x}_t)$ about the target $y_t$. The input $\mathbf{x}_t$ should be considered as a feature vector of the observations before $t-1$, and the prediction $f_{\boldsymbol{\theta}_t}(\mathbf{x}_t)$ at $t$ is about a target $y_t$ that will be observed at step $t$. In online learning, we update $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$ after observing $y_t$ and use $\boldsymbol{\theta}_{t+1}$ to make the prediction, $\hat{y}_{t+1} \equiv f_{\boldsymbol{\theta}_{t+1}}(\mathbf{x}_{t+1})$, about the next target $y_{t+1}$. As we will see, it is straightforward to extend our results to the case of a vector target $\mathbf{y}_t$. We thus assume a scalar target $y_t$ unless otherwise stated.

A standard approach finds the weights $\boldsymbol{\theta}^\star$ that minimize the weighted mean squared error (WMSE)

$$L_t(\boldsymbol{\theta}) \equiv \frac{1}{2} \sum_{d=0}^{t-1} \gamma^d \left(f_{\boldsymbol{\theta}}(\mathbf{x}_{t-d}) - y_{t-d}\right)^2 \tag{1}$$

at step $t$, where the forgetting rate $\gamma$ is a hyperparameter. The optimal weights $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^\star$ are then used to make a prediction about $y_{t+1}$. This loss function (1) is motivated by the expectation that a model recently giving low predictive error gives good prediction for the next pattern. Although minimizing WMSE does not guarantee predictive accuracy in adversarial settings, this objective has been extensively studied in the literature and successfully applied in practice.

For a linear model $f_{\boldsymbol{\theta}}(\mathbf{x}) \equiv \boldsymbol{\theta}^\top \mathbf{x}$, the minimizer of (1) is given by $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}^\star = \mathbf{H}_t^{-1}\, \mathbf{g}_t$, where $\mathbf{g}_t$ is the negative gradient at the origin (*i.e.*, gradient evaluated at $x = 0$) and $\mathbf{H}_t$ is the Hessian of (1), which can be written recursively:

$$\mathbf{g}_t \equiv \sum_{d=0}^{t-1} \gamma^d\, \mathbf{x}_{t-d}\, y_{t-d} = \gamma\, \mathbf{g}_{t-1} + \mathbf{x}_t\, y_t \tag{2}$$

$$\mathbf{H}_t \equiv \sum_{d=0}^{t-1} \gamma^d\, \mathbf{x}_{t-d}\, \mathbf{x}_{t-d}^\top = \gamma\, \mathbf{H}_{t-1} + \mathbf{x}_t\, \mathbf{x}_t^\top. \tag{3}$$

Notice that an alternative to the use of the forgetting rate $\gamma$ is to reset the value of $\mathbf{H}_t$ periodically (Goodwin, Teoh, and Elliott 1983; Moroshko, Vaits, and Crammer 2015), but here we focus on the approaches of using forgetting rates to deal with structural breaks.

By the Sherman-Morrison lemma, one can compute $\mathbf{H}_t^{-1}$ recursively in $O(n^2)$ time[1], where $n$ is the dimension of $\mathbf{x}_t$:

$$\mathbf{H}_t^{-1} = \gamma^{-1}\, \mathbf{H}_{t-1}^{-1} - \frac{\gamma^{-2}\, \mathbf{H}_{t-1}^{-1}\, \mathbf{x}_t\, \mathbf{x}_t^\top\, \mathbf{H}_{t-1}^{-1}}{1 + \gamma^{-1}\, \mathbf{x}_t^\top\, \mathbf{H}_{t-1}^{-1}\, \mathbf{x}_t}. \tag{4}$$

[1]Alternatively, one may recursively update the Cholesky factor of $\mathbf{H}_t$ (Gill et al. 1974), and solve $\mathbf{H}_t\, \boldsymbol{\theta}_{t+1} = \mathbf{g}_t$ for $\boldsymbol{\theta}_{t+1}$.

When $\mathbf{H}_t$ is not invertible, the pseudo-inverse $\mathbf{H}_t^+$ should be used instead of $\mathbf{H}_t^{-1}$. Also, when $\mathbf{H}_{t-1}$ is not invertible in (4), we should use the rank-one update for the pseudo-inverse, which can be performed in $O(n^2)$ time, as follows:

**Lemma 1.** *For a symmetric $\mathbf{H} \in \mathbb{R}^{n \times n}$ and $\mathbf{c} \in \mathbb{R}^n$, let $\mathbf{u} \equiv (\mathbf{I} - \mathbf{H}^+\mathbf{H})\, \mathbf{c}$, $\mathbf{u}^+ \equiv \mathbf{u}/(\mathbf{u}^\top \mathbf{c})$, and $\mathbf{k} \equiv \mathbf{H}^+\mathbf{c}$. Then the pseudo-inverse $(\mathbf{H} + \mathbf{c}\,\mathbf{c}^\top)^+$ can be computed from $\mathbf{H}^+$ as follows: if $\mathbf{u}^\top \mathbf{c} > 0$, then $(\mathbf{H} + \mathbf{c}\,\mathbf{c}^\top)^+ = \mathbf{H}^+ - \mathbf{k}\,(\mathbf{u}^+)^\top - \mathbf{u}^+\mathbf{k}^\top + (1 + \mathbf{c}^\top\mathbf{k})\,\mathbf{u}^+(\mathbf{u}^+)^\top$; if $\mathbf{u}^\top\mathbf{c} = 0$, then $(\mathbf{H} + \mathbf{c}\,\mathbf{c}^\top)^+ = \mathbf{H}^+ - \mathbf{k}\,\mathbf{k}^\top/(1 + \mathbf{c}^\top\mathbf{H}^+\mathbf{c})$.*

Although this lemma directly follows from the rank-one update of the general pseudo-inverse (Theorem 3.1.3 of Campbell and Meyer (2009)), we have made two specific choices for numerical accuracy. First, we define $\mathbf{u}^+$ differently from $\mathbf{u}^+ \equiv \mathbf{u}/||\mathbf{u}||^2$ in Campbell and Meyer (2009), although the two definitions are mathematically equivalent. Second, we define $\mathbf{u}$ as in the lemma, while a mathematically equivalent form of $\mathbf{u} \equiv (\mathbf{I} - \mathbf{H}\,\mathbf{H}^+)\,\mathbf{c}$ is more computationally efficient, as it can use already computed $\mathbf{k}$. Figure 6 in Osogami (2020) shows that these two choices can result in up to $10^{20}$ times higher accuracy.

Because $\mathbf{H}_t$ is independent of the target values, only a single $\mathbf{H}_t^{-1}$ needs to be recursively computed even when there are multiple target values (*e.g.*, vector autoregressive models (Lütkepohl 2005)). The optimal weights are given by $\boldsymbol{\Theta}_t^\star = \mathbf{H}_t^{-1}\, \mathbf{G}_t$ for a recursively computed $\mathbf{G}_t = \gamma\, \mathbf{G}_{t-1} + \boldsymbol{x}_t\, \mathbf{y}_t^\top$ from $\mathbf{G}_0 \equiv \mathbf{O}$.

Our approach may be extended to nonlinear models $f_{\boldsymbol{\theta}}(\cdot)$ by approximation with linearization, analogously to Extended Kalman Filter and the Levenberg-Marquardt method (Anderson and Moore 1979; Nocedal and Wright 2006), which have been successfully applied in practice. However, we continue to study our approach with linear models.

Consider the standard L2 regularized loss function:

$$\dot{L}_t(\boldsymbol{\theta}) = L_t(\boldsymbol{\theta}) + (\lambda/2)\, ||\boldsymbol{\theta}||_2^2. \tag{5}$$

The minimizer of (5) is given by $\dot{\boldsymbol{\theta}}_{t+1} = \dot{\mathbf{H}}_{t+1}^{-1}\, \mathbf{g}_{t+1}$, where $\dot{\mathbf{H}}_t \equiv \mathbf{H}_t + \lambda\, \mathbf{I}$, whose inverse however cannot be updated recursively (Tsakiris 2010; Tsakiris, Lopes, and Nascimento 2010) unless $\lambda = 0$ or $\gamma \in \{0, 1\}$.

We propose the following regularized loss function:

$$\tilde{L}_t(\boldsymbol{\theta}) = L_t(\boldsymbol{\theta}) + (\lambda/2)\, ||\boldsymbol{\theta}||_{\mathbf{H}_t}^2, \tag{6}$$

where $||\boldsymbol{\theta}||_{\mathbf{H}_t}^2 \equiv \boldsymbol{\theta}^\top\, \mathbf{H}_t\, \boldsymbol{\theta}$. The minimizer of $\tilde{L}_t(\cdot)$ is $\tilde{\boldsymbol{\theta}}^\star = \boldsymbol{\theta}^\star/(1 + \lambda)$, where $\boldsymbol{\theta}^\star$ is the minimizer of $L_t(\cdot)$. The magnitude of the weights is thus reduced by a common factor $1 + \lambda$.

It is however a common practice not to regularize the intercept. Let the first element of $\mathbf{x}_t$ be a constant. Then we can write $\mathbf{x}_t^\top = (1, \check{\mathbf{x}}_t^\top)$ and $\boldsymbol{\theta}^\top = (\theta^{(0)}, \check{\boldsymbol{\theta}}^\top)$, where $\theta^{(0)}$ is the intercept. Letting $\hat{\boldsymbol{\theta}}^\top = (0, \check{\boldsymbol{\theta}}^\top)$, we can write our regularized loss function as follows:

$$\tilde{L}_t(\boldsymbol{\theta}) = L_t(\boldsymbol{\theta}) + (\lambda/2)\, ||\hat{\boldsymbol{\theta}}||_{\mathbf{H}_t}^2. \tag{7}$$

The optimal regularized weights can be computed in $O(n^2)$ time, as in the following lemma:

**Lemma 2.** *For linear models, the minimizer of the regularized loss function* (7) *is given by* $\tilde{\boldsymbol{\theta}}^\star = \tilde{\mathbf{H}}_t^{-1}\,\mathbf{g}_t$*, where* $\tilde{\mathbf{H}}_t = \gamma\,\tilde{\mathbf{H}}_{t-1} + \mathbf{x}_t\,\mathbf{x}_t^\top + \lambda\,\hat{\mathbf{x}}_t\,\hat{\mathbf{x}}_t^\top$ *and* $\tilde{\mathbf{H}}_0 = \mathbf{O}$*. Then* $\tilde{\mathbf{H}}_t^{-1}$ *can be computed from* $\mathbf{H}_{t-1}^{-1}$ *in* $O(n^2)$ *time by applying the Sherman-Morrison lemma*[2] *twice.*

*Proof.* We sketch a proof here and provide details in (Osogami 2020). We can rewrite our regularized loss function $\tilde{L}_t(\boldsymbol{\theta})$ using a matrix, $\hat{\mathbf{H}}_t$, which is given by replacing the values of the first row and the first column of $\mathbf{H}_t$ with zeros. Then the minimizer of $\tilde{L}_t(\boldsymbol{\theta})$ can be shown to be given by the $\tilde{\boldsymbol{\theta}}^\star$ in the lemma. The recurrence relation of $\tilde{\mathbf{H}}_t$ follows from the equivalence $\tilde{\mathbf{H}}_t \equiv \mathbf{H}_t + \lambda\,\hat{\mathbf{H}}_t$. □

In addition, our regularized loss function is invariant to invertible linear transformations in the following sense:

**Lemma 3.** *Consider an invertible linear transformation* $\mathbf{M}$ *of order* $n-1$*, and let* $\check{\mathbf{x}}_t' = \mathbf{M}\,\check{\mathbf{x}}_t$ *for each* $t$*. Let the weights except the intercept be contravariate to* $\mathbf{M}$ *in that* $\mathbf{M}$ *transforms* $\check{\boldsymbol{\theta}}^\top$ *into* $\check{\boldsymbol{\theta}}'^\top = \check{\boldsymbol{\theta}}^\top\mathbf{M}^{-1}$*. Then the loss function* (7) *is invariant to* $\mathbf{M}$*.*

*Proof.* We sketch a proof here and provide details in (Osogami 2020). It is known and straightforward to show that $L_t(\cdot)$ is invariant to $\mathbf{M}$. It thus remains to prove that the regularization term is invariant to $\mathbf{M}$. It can be shown that $\mathbf{M}$ transforms $||\hat{\boldsymbol{\theta}}||^2_{\mathbf{H}_t}$ into $||\hat{\boldsymbol{\theta}}'||^2_{\mathbf{H}_t'} = \check{\boldsymbol{\theta}}'^\top\check{\mathbf{H}}_t'\check{\boldsymbol{\theta}}'$, where $\check{\mathbf{H}}_t' = \mathbf{M}\,\check{\mathbf{H}}_t\,\mathbf{M}^\top$. Thus, $||\hat{\boldsymbol{\theta}}'||^2_{\mathbf{H}_t'} = \check{\boldsymbol{\theta}}^\top\,\check{\mathbf{H}}_t\,\check{\boldsymbol{\theta}} = ||\hat{\boldsymbol{\theta}}||^2_{\mathbf{H}_t}$. □

This invariance is in contrast to L2 regularization, which is sensitive to the transformation of the coordinates of explanatory variables. See Figure 4 in Osogami (2020).

Note that, although it is new to use $||\boldsymbol{\theta}||^2_{\mathbf{H}_t}$ as a regularizer of recursive least squares, the norm $||\cdot||^2_{\mathbf{H}_t}$ has been used in various contexts in the literature. For example, a similar norm has been used in the analysis of linear stochastic bandits by Abbasi-yadkori, Pál, and Szepesvári (2011).

## Following Best Hyper Forgetting Rate

We have seen that the optimal parameters can be updated in $O(n^2)$ time at each step for given values of hyperparameters, $\gamma$ and $\lambda$. We now study what the good values of $\gamma$ and $\lambda$ are and how to adaptively choose those values in $O(n^2)$ time at each step.

Our hyperparameters determine the loss function, which is designed with the expectation that its minimizer gives minimal predictive error at each step. The good values of $\gamma$ and $\lambda$ are the ones that lead to good prediction, but they can change over time due to structural breaks. We thus need to tune those values adaptively. A difficulty is that $\mathbf{g}_t$ and $\tilde{\mathbf{H}}_t$ depend on $\gamma$ or $\lambda$. When the values of $\gamma$ or $\lambda$ change, we need to recompute $\mathbf{g}_t$ and $\tilde{\mathbf{H}}_t$, which requires $\Omega(n^2)$ time.

We propose to train a small number $N_{\mathrm{mod}}$ of the models in a way that they minimize the loss functions with varying values of the forgetting rate $\gamma_i$ and regularization-coefficient $\lambda_i$

[2]We use Lemma 1 instead of the Sherman-Morrison lemma, when the inverse under consideration does not exist.

**Algorithm 1** Online learning by following the best hyper forgetting rate (single target)

**Input:** $N_{\mathrm{mod}} = 30$, $N_{\mathrm{hyp}} = 11$; $\gamma_1 = \lambda_1 = 0$, $\gamma_i \sim \mathrm{Unif}[0.5^{1/D}, 1]$, $\lambda_i \sim \mathrm{Unif}[0,1], \forall i \in [2, N_{\mathrm{mod}}]$; $\eta_j = 0.89 + 0.01\,j, \forall j \in [1, N_{\mathrm{hyp}}]$
$\mathrm{CSE}(\eta_j), \mathrm{CSE}^{(i)}(\eta_j) \leftarrow 0, 0, \forall i, j$
$\mathbf{g}^{(i)}, (\mathbf{H}^{(i)})^{-1} \leftarrow \mathbf{0}, \mathbf{O}, \forall i$
**for** $t = 1, 2, \ldots$ **do**
  Prepare a feature vector $\mathbf{x}_t$
  $j^\star = \mathrm{argmin}_j \mathrm{CSE}(\eta_j)$
  $i^\star \leftarrow \mathrm{argmin}_i \mathrm{CSE}^{(i)}(\eta_{j^\star})$
  Output prediction: $\hat{y}_t^{(i^\star)} \leftarrow \mathbf{x}_t^\top(\mathbf{H}^{(i^\star)})^{-1}\mathbf{g}^{(i^\star)}$
  Observe the target $y_t$
  **for** $j = 1, \ldots, N_{\mathrm{hyp}}$ **do**
    $i^\star \leftarrow \mathrm{argmin}_i \mathrm{CSE}^{(i)}(\eta_j)$
    $\mathrm{CSE}(\eta_j) \leftarrow \mathrm{CSE}(\eta_j) + (\hat{y}^{(i^\star)} - y_t)^2$
  **end for**
  **for** $i = 1, \ldots, N_{\mathrm{mod}}$ **do**
    $\hat{y}_t^{(i)} \leftarrow \mathbf{x}_t^\top(\mathbf{H}^{(i)})^{-1}\mathbf{g}^{(i)}$
    $\mathrm{CSE}^{(i)}(\eta_j) \leftarrow \eta_j \mathrm{CSE}^{(i)}(\eta_j) + (\hat{y}_t^{(i)} - y_t)^2, \forall j$
    $\mathbf{g}^{(i)} \leftarrow \mathbf{g}^{(i)}/\gamma^{(i)} + \mathbf{x}_t\,y_t$
    $(\mathbf{H}^{(i)})^{-1} \leftarrow (\mathbf{H}^{(i)})^{-1}/\gamma^{(i)} + \mathbf{x}_t\,\mathbf{x}_t^\top$
    $(\mathbf{H}^{(i)})^{-1} \leftarrow \mathtt{Update}((\mathbf{H}^{(i)})^{-1}, \mathbf{x}_t)$
    $(\mathbf{H}^{(i)})^{-1} \leftarrow \mathtt{Update}((\mathbf{H}^{(i)})^{-1}, \sqrt{\lambda^{(i)}}\,\hat{\mathbf{x}}_t)$
  **end for**
**end for**

(see Figure 1), which are randomly selected for each model $i$ (see Algorithm 1 for the specific distributions that we recommend). We then track the cumulative squared error of the prediction, $\hat{y}_t^{(i)}$, given by each model $i$, where the error is discounted by a hyper forgetting rate $\eta \in (0,1)$:

$$\mathrm{CSE}_t^{(i)}(\eta) \equiv \sum_{d=0}^{t-1} \eta^d\,(\hat{y}_{t-d}^{(i)} - y_{t-d})^2 \tag{8}$$

$$= \eta\,\mathrm{CSE}_{t-1}^{(i)} + (\hat{y}_t^{(i)} - y_t)^2. \tag{9}$$

The model $i$ having the smallest CSE has been giving small error, and such a model $i$ is expected to give good prediction in the next step. Although we can now adaptively choose the values of $\gamma$ and $\lambda$ to be the ones that minimize (9), we now need to tune $\eta$, whose suitable value can change over time.

We use multiple values of the hyper forgetting rate $\eta_1, \eta_2, \ldots$ (see Figure 1; also see Algorithm 1 for the specific values we recommend). We track $\mathrm{CSE}_t^{(i)}(\eta_j)$ for each $\eta_j$. If we evaluated the models with $\eta_j$, we would choose the model $i_{t+1}^\star = \mathrm{argmin}_i \mathrm{CSE}_t^{(i)}(\eta_j)$ as the best performing model at $t+1$. We thus evaluate the performance of a hyper forgetting rate $\eta_j$ via

$$\mathrm{CSE}_t(\eta_j) \equiv \sum_{d=0}^{t-1} (\hat{y}_{t-d}^{(i_{t-d}^\star)} - y_{t-d})^2 \tag{10}$$

$$= \mathrm{CSE}_{t-1}(\eta_j) + (\hat{y}_t^{(i_t^\star)} - y_t)^2, \tag{11}$$

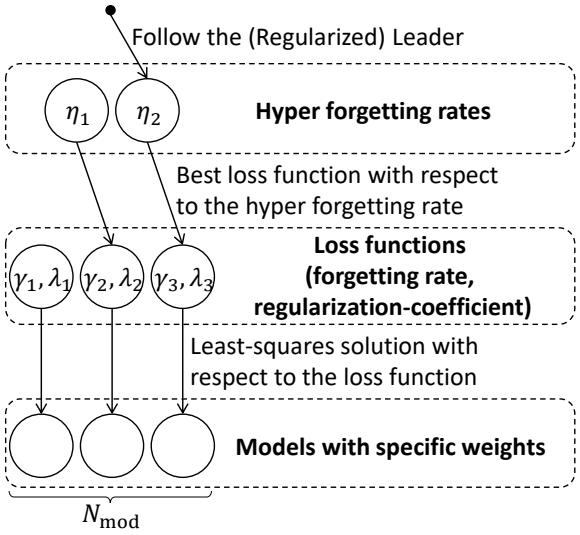



Figure 1: Following the best hyper forgetting rate. FTL chooses the best hyper forgetting rate, which in turn chooses the best loss function, which then uniquely determines the model with specific weights as the least-squares solution.

where recall that $\hat{y}_t^{(i_t^\star)}$ is the prediction by the best performing model with respect to the $\mathrm{CSE}_{t-1}^{(\cdot)}(\eta_j)$. Namely, the best performing hyper forgetting rate at step $t$ is selected by $j_t^\star = \mathrm{argmin}_j \mathrm{CSE}_t(\eta_j)$, which is then used to select the best performing model at step $t$ by $i_{t+1}^\star = \mathrm{argmin}_i \mathrm{CSE}_t^{(i)}(\eta_{j_t^\star})$. with $j \equiv j_t^\star$. More specifically, at step $t+1$, the minimizer $i_{t+1}^\star$ defines our loss function (7) with $\gamma = \gamma^{(i_{t+1}^\star)}$ and $\lambda = \lambda^{(i_{t+1}^\star)}$. Then the minimizer $\boldsymbol{\theta}^\star \equiv (\tilde{\mathbf{H}}_t^{(i_{t+1}^\star)})^{-1}\, \mathbf{g}_t^{(i_{t+1}^\star)}$ of this loss function is used to give the prediction $\mathbf{x}_t^\top\, \boldsymbol{\theta}^\star$ about step $t+1$. We recursively compute $(\tilde{\mathbf{H}}_t^{(i)})^{-1}$ and $\mathbf{g}_t^{(i)}$ for every $i$ at every step even if they may not be needed.

Notice that the hyper forgetting rate gives a criteria to determine what forgetting rates are good. One might want to introduce a "hyper-hyper forgetting rate" in (10) to determine what hyper forgetting rates are good, and this can be continued indefinitely. Our method can be considered as a first order approximation to such sequential argument.

At each step $t$, Algorithm 1 first chooses the best hyper forgetting rate $\eta_j^\star$ in Step 6 and then uniquely maps an input (past values of time-series) to an output (prediction) in Step 7-8 based on the $\eta_j^\star$. Hence, the hyper forgetting rate may be considered as a model in the context of Follow The Leader (FTL) or more generally in the context of online learning.

One may thus use other online learning algorithms (Hazan 2016) instead of FTL. For example, it is straightforward to replace FTL with Follow the Regularized Leader (FoReL). Specifically, at each step $t$, we find the probability vector $\mathbf{p}^\star$, which we sample a hyper forgetting rate from:

$$\mathbf{p}^\star = \operatorname*{argmin}_{\mathbf{p}} \sum_j p_j\, \mathrm{CSE}_t(\eta_j) + \alpha \sum_j p_j\, \ln p_j. \qquad (12)$$

The following regret bound is then a straightforward consequence of Corollary 2.14 from Shalev-Shwartz (2011):

**Corollary 1.** *Let $f_s(\mathbf{p})$ be the expected squared error in prediction at step $s$ when the hyper forgetting rate is chosen with probability $\mathbf{p}$ in Step 6 of Algorithm 1. Let $L_s$ and $L$ be such that $f_s$ is $L_s$-Lipschitz with respect to L1 norm and $\frac{1}{t}\sum_{s=1}^t L_s^2 \le L^2$. By setting $\alpha = L\sqrt{2\,t/\ln N_{\mathrm{hyp}}}$ in (12), the expected regret (against the best static probability) for the cumulative squared error can be bounded by* $\mathrm{Regret}(t) \le L\,\sqrt{2\,\ln(N_{\mathrm{hyp}})\,t}$.

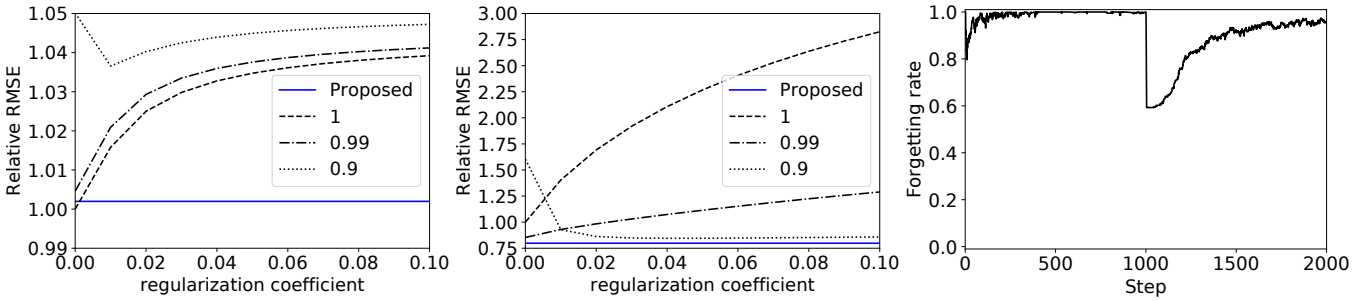
An alternative to our approach is to consider a convex combination of the outputs from the $N_{\mathrm{mod}}$ models (Andersson 1985; Arenas-Garcia, Figueiras-Vidal, and Sayed 2006) rather than choosing a single model. For example, Arenas-Garcia, Figueiras-Vidal, and Sayed (2006) updates the weight of the convex combination with a gradient based approach. The effectiveness of such approaches relies on optimizing the learning rate in the gradient based approach, but it is not well understood how best to choose the learning rate. Our approach of hyper forgetting rates may also be applied to optimizing the learning rate in this context.

Algorithm 1 summarizes our algorithm. Here, the subscript $t$ is dropped from $\mathbf{g}_t^{(i)}$, $\mathbf{H}_t^{(i)}$, $\mathrm{CSE}_t$, and $\mathrm{CSE}_t^{(i)}$; $(\mathbf{H}^{(i)})^{-1}$ denotes the pseudo-inverse when $\mathbf{H}^{(i)}$ is not invertible. **Input** specifies the values of the hyperparameters for each model $i$, where "$\sim \mathrm{Unif}[a,b]$" means that a sample is independently drawn from the uniform distribution with support $[a,b]$. Here, $D$ determines this support. Intuitively, if a model takes into account long history, one should not forget the past by a large amount. For an autoregressive model, $D$ may be its order. Step 5 depends on the time-series model under consideration. For example, $\mathbf{x}_t$ is the concatenation of $d$ previous observations and a constant 1 for an autoregressive model with order $d$. The for-loop from Step 10 updates the CSE for each hyper forgetting rate. In the for-loop from Step 14, the weights are updated for each $i$, following the approach given in the previous section (Regularized recursive least squares). `Update` in Step 19-20 applies (4) when $\mathbf{H}^{(i)}$ is invertible and Lemma 1 otherwise.

Steps 6-7, 10-13, and 16 as well as the for-loop from Step 14 are the additional steps for adaptively choosing the values of $\gamma$ and $\lambda$. The computational complexity of these additional steps depends on $N_{\mathrm{mod}}$ and $N_{\mathrm{hyp}}$ but is independent of $n$ and $t$. Algorithm 1 thus runs in $O(n^2)$ time per step. When there are multiple target values, we track the CSE for each target and choose the optimal model for each target. The computational complexity of the associated steps thus grows linearly with respect to the dimension of the target, which we assume $O(n)$. Because these steps have $O(n)$ complexity per step for the case of a single target, the overall complexity per step remains $O(n^2)$.

## Experiments

We conduct numerical experiments to answer the following questions. (i) How does our regularization compare against the standard L2 regularization? (ii) Can Algorithm 1 adaptively tune hyperparameters? (iii) How does our approach compare against existing methods in predicting nonstationary time-series in (financial) industries? We have designed three sets of experiments, corresponding to these questions.




(a) (b) (c)

Figure 2: The results of the experiments with the synthetic time-series with a change point at $t = 1,000$ (averaged over 30 runs): RMSE during the 100 steps immediately (a) before and (b) after the change point; (c) the forgetting rate used by Algorithm 1 at each step. The regularization-coefficient used by Algorithm 1 at each step is shown in Figure 7 of Osogami (2020).

All of our experiments are in a setting of online learning: for a time-series of length $N$, we make a prediction about the next value at every step $n$ for $0 < n < N$. When we make a prediction at step $n$, the time-series up to step $n$ is used to train the models. The RMSE will refer to the root mean squared error of the predictions over the period under consideration. We run our experiments on a workstation having eight Intel Core i7-6700K CPUs running at 4.00 GHz and 64 GB random access memory.

In Figure 5 of Osogami (2020), we compare the effectiveness of our regularization against L2 regularization. Overall, our regularization compares favorably against L2 regularization. Although the effectiveness of regularization depends on particular data, the results of this experiment suggest that our regularization not only can be performed in $O(n^2)$ time but also has the expected effect of regularization, sometimes outperforming L2 regularization (*e.g.* when a time-series involves large fluctuations).

In Figure 2, we study the next question of whether Algorithm 1 can adaptively tune the forgetting rate $\gamma$ and regularization-coefficient $\lambda$. Here, we use synthetic time-series with a change point, similar to Miyaguchi and Kajino (2019), and study how our algorithm behaves at this change point. Specifically, we generate a time-series of 2,000 steps according to an auto-regressive (AR) model, which is a particular linear model: $x_t = \mu_t + a_t\, x_{t-1} + \varepsilon_t$, where $\varepsilon_t$ is i.i.d. with the standard normal distribution. We set $(\mu_t, a_t) = (-10, 0.3)$ for $t < 1,000$, and $(\mu_t, a_t) = (10, -0.3)$ for $t \geq 1,000$. The change point is $t = 1,000$. We learn the AR model with the first order with Algorithm 1, and compares its predictive error against the baseline where the values of $\gamma$ and $\lambda$ are fixed.

Figure 2 (a)-(b) show the RMSE of the prediction during the 100 steps immediately (a) before and (b) after the change point. The straight lines indicate the RMSE with Algorithm 1, where $\gamma$ and $\lambda$ are adaptively tuned. The three curves in each panel show the RMSE of the prediction for each value of $\gamma$ (as indicated in the legend) and for each value of $\lambda$ (as indicated along the horizontal axis). Before the change (*i.e.*, in a stationary period), setting $\gamma = 1$ and $\lambda = 0$ works best and outperforms Algorithm 1. After the change (*i.e.*, in a nonstationary period), the optimal setting for the stationary period has quite poor performance, and the setting with low $\gamma$ and high $\lambda$ works best. Overall, Algorithm 1 performs as good as the best performing static choice of $\gamma$ and $\lambda$ for each period.

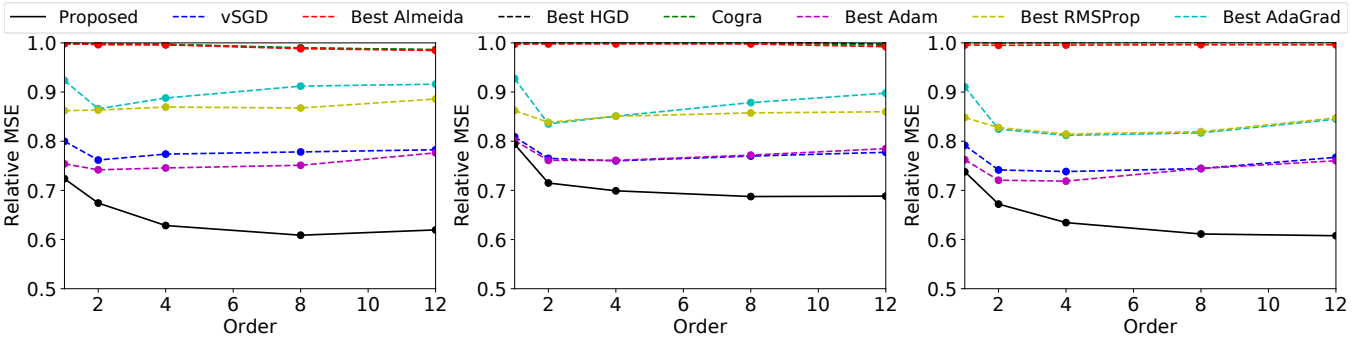
Figure 2 (c) and Figure 7 of Osogami (2020) show the values of the $\gamma$ and $\lambda$ used by Algorithm 1 at each step. In the beginning of the time-series, our algorithm uses a relatively small $\gamma$, and a relatively large $\lambda$. We can see that these values converge to $\gamma = 1$ and $\lambda = 0$ if the time-series is stationary for a sufficiently long period. Immediately after the change point at $t = 1,000$, Algorithm 1 drops the forgetting rate at $\gamma \approx 0.6$ and increases the regularization-coefficient at $\lambda \approx 0.4$.

Finally, we apply Algorithm 1 to financial time-series and compares its predictive error against existing machine learning methods for nonstationary time-series. We use the 10-year (from September 1, 2008 to August 31, 2018) historical data of the daily close price of Standard & Poor's 500 Stock Index (US index; SPX), Nikkei 225 (Japanese index; Nikkei 225), Deutscher Aktienindex (German index; DAX), Financial Times Stock Exchange 100 Index (UK index; FTSE 100), and Shanghai Stock Exchange Composite Index (Chinese index; SSEC). We choose these indices, because they are representative stock indices around the world.

Each time-series $x_t$ is converted into the absolute value of the daily return $r_t \equiv |x_t - x_{t-1}|/x_{t-1}$, and we predict the difference from the previous day (*i.e.*, predict $r_{t+1} - r_t$ at day $t$). The absolute return is related to the volatility, and its prediction is useful particularly for risk-sensitive trading. In addition, the absolute return $r_t$ is more predictable than the raw price $x_t$ and hence is more suitable for studying relative accuracy of various predictors.

Figure 3 shows the mean squared error (MSE) of the prediction by Algorithm 1 and baselines: vSGD, Almeida, Hypergradient descent (HGD), Cogra, Adam, RMSProp, and AdaGrad. The MSE is normalized relative to that of the naïve prediction that the absolute daily return stays unchanged from the previous day. The solid curves show the relative MSE of Algorithm 1. The horizontal axis represents the order of the AR model that we train [3]. Because

[3] Algorithm 1 uses AR models with smaller order during the




(a) SPX (b) Nikkei 225 (c) DAX

Figure 3: MSE of the predicted absolute daily return of various financial indices (as indicated in each column), relative to the naïve prediction of no change. Algorithm 1 (solid curve) is compared against seven baselines with the optimal choice of hyperparameters (dashed curves). Results with FTSE 100 and SSEC are shown in Figure 12 of Osogami (2020). Error bars are shown in Figure 8-12 of Osogami (2020).

the baselines except vSGD and Cogra have hyperparameters that cannot be automatically tuned, the best performing one (on the test data) is shown[4].

Overall, Algorithm 1 gives significantly smaller predictive error (see Figures 8-12 in Osogami (2020) for error bars) than any of the baselines for all financial indices under consideration, even when the hyperparameters of the baselines are set optimally with the knowledge of the entire time-series in advance. In SPX, the relative MSE of 0.608 is achieved (at order 8) by Algorithm 1, while the best performing baseline (Adam) achieves MSE of 0.741 (at order 2), which is 22% higher than our MSE.

Algorithm 1 outperforms baselines primarily because it minimizes (1) with the hyperparameters that minimize (9) for the hyper forgetting rate that minimizes (11) at every point in time. Baselines do not minimize these quantities. Our experiments suggest that minimizing (1), (9), and (11) leads to good predictive performance for the tasks under consideration, although it does not imply that the proposed method outperforms baselines for all forecasting tasks.

Because Algorithm 1 trains $N_{\text{mod}} = 30$ models in parallel, it requires more computational cost than baselines. For SPX, Algorithm 1 with order 12 requires 7.1 seconds, which is approximately $N_{\text{mod}}$ times slower than the baselines (vSGD: 0.77, Cogra: 0.87, AdaGrad: 0.26, HGD: 0.40, Almeida: 0.33, RMSProp: 0.27, and Adam: 0.31, where the units are all in seconds)[5]. The computational cost associated with inverse Hessian in Algorithm 1 is negligible for this size of time-series.

Algorithm 1 is relatively robust against the small changes in the values of these hyperparameters. For example, we have run experiments after adding an $\eta_j = 0$ in Algorithm 1. However, the added $\eta_j = 0$ is hardly ever selected as the best, and the overall behavior of Algorithm 1 stays essentially unchanged.

## Conclusion

Our experiments confirm that (i) our regularization can outperform L2 regularization for some time-series with large fluctuations, (ii) Algorithm 1 can quickly tune hyperparameters after a change point, and (iii) our approach outperforms baselines in predicting financial time-series. These answer favorably to our questions posed in the beginning of the previous section (Experiments).

Our approach involves two key techniques: transformation-invariant regularization and "follow the best hyper forgetting rate." In this paper, we have demonstrated two advantages of transformation-invariant regularization over L2 regularization: it (i) allows us to update the inverse Hessian in $O(n^2)$ time and (ii) has the expected effect of regularization for the time-series with large fluctuation. Our "follow the best hyper forgetting rate" method uses the Follow The Leader method (or other online learning algorithms such as Follow the Regularized Leader, as discussed) to select a hyper forgetting rate, which in turn uniquely (after tie-break) determines a model to make a prediction. The hyper forgetting rate is used to discount the predictive error made by the model (with specific values of the forgetting rate and regularization-coefficient) in the past. By selecting the hyper forgetting rate, instead of directly selecting a model without hyper forgetting or with an arbitrarily set hyper forgetting rate, our approach can better evaluate the model under nonstationarity, which in turn allows us to select a better model at each step.

initial period. Specifically, order $d$ is used during the period $[2(d+1)^2/(d+1), 2(d+2)^2/(d+2))$.

[4]We vary the initial learning rate (and the hyper learning rate of Almeida and HGD) in $\{1, 0.1, 0.01, 0.001, 0.0001\}$ and the hyperparameters related to "forgetting" (two parameters in Adam; one in RMSProp) in $\{0.9, 0.99, 0.99, 0.9999\}$. For each data point, the best test performance among all of the possible combinations of those hyperparameters is shown, so that the error reported for the baselines is generally smaller than what can be achieved without the knowledge of the test data.

[5]The ground total running time (seconds) with all configurations of hyperparameters under consideration: 1.3 for AdaGrad; 2.0 for HGD; 1.6 for Almeida; 5.4 for RMSProp; 24.8 for Adam.

## Acknowledgments

Earlier versions of this work were supported by JST CREST Grant Number JPMJCR1304, Japan.

## References

Abbasi-yadkori, Y.; Pál, D.; and Szepesvári, C. 2011. Improved Algorithms for Linear Stochastic Bandits. In Shawe-Taylor, J.; Zemel, R. S.; Bartlett, P. L.; Pereira, F.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 24*, 2312–2320. Curran Associates, Inc. URL http://papers.nips.cc/paper/4417-improved-algorithms-for-linear-stochastic-bandits.pdf.

Agarwal, A.; Hazan, E.; Kale, S.; and Schapire, R. E. 2006. Algorithms for portfolio management based on the Newton method. In *Proceedings of the 23rd International Conference on Machine Learning*, 9–16.

Almeida, L. B.; Langlois, T.; Amaral, J. D.; and Plakhov, A. 1999. Parameter adaptation in stochastic optimization. In *On-Line Learning in Neural Networks*, chapter 6, 111–134. Cambridge University Press.

Anderson, B. D. O.; and Moore, J. B. 1979. *Optimal Filtering*. Prentice-Hall.

Andersson, P. 1985. Adaptive forgetting in recursive identification through multiple models. *International Journal of Control* 42(5): 1175–1193.

Arenas-Garcia, J.; Figueiras-Vidal, A. R.; and Sayed, A. H. 2006. Mean-square performance of a convex combination of two adaptive filters. *IEEE Transactions on Signal Processing* 54(3): 1078–1090.

Baydin, A. G.; Cornish, R.; Rubio, D. M.; Schmidt, M.; and Wood, F. 2017. Online learning rate adaptation with hypergradient descent. *CoRR* abs/1703.0478.

Campbell, S. L.; and Meyer, C. D. 2009. *Generalized Inverses of Linear Transformations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics.

De Rooij, S.; Van Erven, T.; Grünwald, P. D.; and Koolen, W. M. 2014. Follow the leader if you can, hedge if you must. *The Journal of Machine Learning Research* 15(1): 1281–1316.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12: 2121–2159.

Elisei-Iliescu, C.; Stanciu, C.; Paleologu, C.; Benesty, J.; Anghel, C.; and Ciochină, S. 2017. Robust variable-regularized RLS algorithms. In *Proceedings of the 2017 Hands-free Speech Communications and Microphone Arrays*, 171–175.

Farhang-Boroujeny, B. 2013. *Adaptive Filters: Theory and Applications*. Wiley, 2nd edition.

Freund, Y.; and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55: 119–139.

Garrigues, P.; and Ghaoui, L. E. 2009. An Homotopy Algorithm for the Lasso with Online Observations. In *Advances in Neural Information Processing Systems 21*, 489–496. Curran Associates, Inc.

Gay, S. L. 1996. Dynamically regularized fast RLS with application to echo cancellation. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 957–960.

Gill, P.; Golub, G.; Murray, W.; and Saunders, M. 1974. Methods for modifying matrix factor-izations. *Mathematics of Computation* 126(28): 505–535.

Goodwin, G. C.; Teoh, E. K.; and Elliott, H. 1983. Deterministic convergence of a self-tuning regulator with covariance resetting. *Control Theory and Applications* 130(1): 6–8.

Granger, C. W. J.; and Newbold, P. 1986. *Forecasting Economic Time Series*. Economic Theory, Econometrics, and Mathematical Economics. Academic Press, Inc., 2nd edition.

Guo, L.; Ljung, L.; and Priouret, P. 1993. Performance analysis of the forgetting factor RLS algorithm. *International Journal of Adaptive Control and Signal Processeing* 7: 525–537.

Hazan, E. 2016. *Introduction to Online Convex Optimization*. Now Pub.

Hazan, E.; Agarwal, A.; and Kale, S. 2007. Logarithmic regret algorithms for online convex optimization. *Machine Learning* 69(2-3): 169–192.

Hazan, E.; and Seshadhri, C. 2009. Efficient learning algorithms for changing environments. In *Proceedings of the 26th International Conference on Machine Learning*, 393–400.

Horita, E.; Sumiya, K.; Urakami, H.; and Mitsuishi, S. 2004. A Leaky RLS Algorithm: Its Optimality and Implementation. *IEEE Transactions on Signal Processing* 52(10): 2924–2936.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980.

Lafond, J.; Wai, H.-T.; and Moulines, E. 2016. On the Online Frank-Wolfe Algorithms for Convex and Non-convex Optimizations. *CoRR* abs/1510.01171.

Littlestone, N.; and Warmuth, M. 1994. The Weighted Majority Algorithm. *Information and Computation* 108: 212–261.

Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. Springer-Verlag Berlin Heidelberg.

Minku, L. L. 2019. A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering* 24: 3153–3204.

Miyaguchi, K.; and Kajino, H. 2019. Cogra: Concept-drift-aware Stochastic Gradient Descent for Time-series Forecasting. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)*.

Moroshko, E.; Vaits, N.; and Crammer, K. 2015. Second-order non-stationary online learning for regression. *The Journal of Machine Learning Research* 16: 1481–1517.

Nocedal, J.; and Wright, S. 2006. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition.

Osogami, T. 2020. Proofs and additional experiments on Second order techniques for learning time-series with structural breaks. *CoRR* abs/2012.08037.

Schaul, T.; Zhang, S.; and LeCun, Y. 2013. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013), PMLR 28(3)*, 343–351.

Shalev-Shwartz, S. 2011. Online learning and online convex optimization. *Foundations and Trends in Machine Learning* 4(2): 107–194.

Tieleman, T.; and Hinton, G. E. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Tsakiris, M. 2010. *On the regularization of the recursive least-squares algorithm*. Master's thesis, Escola Politécnica.

Tsakiris, M. C.; Lopes, C. G.; and Nascimento, V. H. 2010. An array recursive least-squares algorithm with generic non-fading regularization matrix. *IEEE Signal Processing Letters* .

van Erven, T.; and Koolen, W. M. 2016. MetaGrad: Multiple Learning Rates in Online Learning. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 3666–3674. Curran Associates, Inc.

Van Vaerenbergh, S.; Santamaría, I.; and Lázaro-Gredilla, M. 2012. Estimation of the forgetting factor in kernel recursive least squares. In *2012 IEEE International Workshop on Machine Learning for Signal Processing*.