Statistics estimation in neural network training: a recursive identification approach

Ruth Crasto [1] **Xuchan Bao** [1] **Roger Grosse** [1]

Abstract

A common practice in mini-batch neural network training is to estimate global statistics using exponential moving averages (EMA). However, such methods can be sensitive to the EMA decay parameter, which is typically set by hand. In this paper, we introduce **A**daptive **Li**near **S**tate **E**stimation (ALiSE), an online method for adapting the parameters of a linear estimation model such as an EMA. Our work establishes a connection between parameter estimation methods in deep learning, including ALiSE, and recursive identification techniques in control theory. We apply ALiSE to a range of deep learning scenarios and show that it can learn sensible schedules for the EMA decay parameter. Compared to the naive EMA baseline, ALiSE leads to matching or accelerated convergence during training.

1. Introduction

When training a neural network, the optimization update for the network weights often requires an estimate of statistics from noisy observations. The statistics may appear as part of the optimization algorithm, or as part of the training objective. For example, the Adam optimizer (Kingma & Ba, 2014) requires estimating the first and second moments of the gradient.

When the update direction is a nonlinear function of the statistics, using noisy statistics estimates will introduce bias in the update, which can lead to slower convergence. A common practice is to instead estimate the statistics with an exponential moving average (EMA), using a constant hand-tuned decay parameter between 0 and 1. While using EMA can eventually reduce the bias in the update, performance in the earlier stages of training may be sensitive to the value of the decay parameter. A decay parameter that is too small results in noisy estimates, while a decay parameter too close to 1 is unable to track fast-changing statistics. Moreover, the best choice of decay parameter likely changes over the course of training, as the true statistics may evolve at different rates over time. This makes a constant, hand-tuned decay parameter inadequate.

In this paper, we propose ALiSE (**A**daptive **Li**near **S**tate **E**stimation), a method for adapting the parameters of a linear estimation model for statistics estimation in neural network training. In particular, ALiSE can be used to adapt the decay parameter in an EMA model. Our contributions are summarized below:

1. We establish a connection between online parameter estimation techniques in deep learning and recursive identification. Specifically, we show that many of these are examples of recursive prediction-error methods. A unified view of online parameter estimation inspired by recursive identification can provide insight into i) the conditions necessary for these methods to perform well, and ii) heuristics for tuning their hyperparameters.

2. We propose ALiSE, a direct application of the recursive prediction-error framework to a linear state estimation model. We apply ALiSE to various online statistics estimation problems in deep learning. ALiSE can recover sensible parameter schedules that match or exceed performance of the best naive EMA baselines.

2. Background

Let $f(w_t, z) \in \mathbb{R}^n$ be a function of an input data point z and neural network weights w at time t. We use $\mathrm{x}_t = \mathbb{E}_z[f(w_t, z)]$ to denote the true time-varying statistics that we would like to estimate. The true statistics x_t is the expected value of $f(w_t, z)$ over the entire data distribution. Since this paper takes a state estimation approach for estimating the statistics, we refer to x as the "state", even though it may not contain enough information to predict the future evolution of the system.

[1]Department of Computer Science, University of Toronto, Toronto, Canada. Correspondence to: Ruth Crasto <ruth.crasto@mail.utoronto.ca>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

2.1. Statistics estimation using EMA

Consider the following optimization update when training a neural network:

$$w_{t+1} \leftarrow w_t - \alpha_t g(w_t, \mathrm{B}_t, \hat{\mathrm{x}}_t) \tag{1}$$

where α_t is the learning rate, and $g(w_t, \mathrm{B}_t, \hat{\mathrm{x}}_t)$ is a gradient-like signal[1] that is a function of the network weights w_t, the current mini-batch B_t, and an estimate of the statistics $\hat{\mathrm{x}}_t$. This is a common setup in machine learning. For example, the Adam optimizer (Kingma & Ba, 2014) computes an update in the form of (1), where the statistics vector x_t contains the first and second moments of the gradient. More examples are provided in Section 6.

Though the true statistics x_t involves an expectation over the entire data distribution, in (1) we only have access to the mini-batch sample of the data B_t. A naive approach is to replace the expectation with the average over the mini-batch data, denoted as $\hat{\mathrm{x}}_t^{\mathrm{B}}$. Assuming the mini-batch data is sampled i.i.d. from the data distribution, $\hat{\mathrm{x}}_t^{\mathrm{B}}$ is an unbiased estimate of the true statistics x_t:

$$\mathbb{E}[\hat{\mathrm{x}}_t^{\mathrm{B}}] = \mathrm{x}_t, \quad \text{where} \quad \hat{\mathrm{x}}_t^{\mathrm{B}} = \frac{1}{|\mathrm{B_t}|} \sum_{z \in \mathrm{B}_t} f(w_t, z)$$

However, if g is a nonlinear function of x_t, using the mini-batch estimate will result in a biased gradient update, which is undesirable for optimization:

$$\mathbb{E}[g(w_t, \mathrm{B}_t, \hat{\mathrm{x}}_t^{\mathrm{B}})] \neq g(w_t, \mathrm{B}_t, \mathbb{E}[\hat{\mathrm{x}}_t^{\mathrm{B}}]) = g(w_t, \mathrm{B}_t, \mathrm{x}_t)$$

A simple approach that can potentially reduce the bias is to apply an exponential moving average (EMA) on the noisy statistics estimates, where $\beta \in [0, 1)$ is the decay parameter:

$$\hat{\mathrm{x}}_{t+1} = \beta \hat{\mathrm{x}}_t + (1 - \beta) \mathrm{y}_t,$$

For notational consistency with the system identification literature, we use y_t to denote a noisy observation of the statistics (e.g. a mini-batch estimate $\hat{\mathrm{x}}_t^{\mathrm{B}}$). At steady-state (meaning $\hat{\mathrm{x}}_t$ and $\hat{\mathrm{x}}_{t+1}$ have the same variance), EMA can reduce the variance of the estimate $\hat{\mathrm{x}}$ by a factor of $\frac{1-\beta}{1+\beta}$, assuming $\hat{\mathrm{x}}_t$ and y_t are independent. However, away from the steady-state (before training converges), the training performance can be sensitive to the decay parameter. β being too small leads to noisy estimates $\hat{\mathrm{x}}_t$, which in turn causes a larger amount of bias in the gradient update. β being too close to 1 would cause a significant lag between the estimate $\hat{\mathrm{x}}_t$ and the true statistics x_t.

The optimal choice of β depends on how fast the statistics evolves at a given time, which likely changes over the course of training. This makes a hand-tuned, constant β value inadequate. We need an estimation model whose parameters are jointly updated with the state estimate, $\hat{\mathrm{x}}$.

[1] $g(w_t, \mathrm{B}_t, \mathrm{x}_t)$ can be either the gradient itself, or some modifications of the gradient (e.g. with a different search direction).

2.2. Linear state estimation model

In this paper, we will consider a linear estimation model of the state $\hat{\mathrm{x}}$, where $A, B : \mathbb{R}^d \rightarrow \mathbb{R}^{n \times n}$ are matrices parameterized by θ, and $\hat{\mathrm{y}}$ is the prediction:[2]

$$\hat{\mathrm{x}}_{t+1} = A(\theta)\hat{\mathrm{x}}_t + B(\theta)\mathrm{y}_t \tag{2a}$$

$$\hat{\mathrm{y}}_t = \hat{\mathrm{x}}_t \tag{2b}$$

Note that an EMA with scalar decay coefficient β is a special case of (2) with $d = 1$, $A(\beta) = \beta I$, and $B(\beta) = (1 - \beta)I$.

3. Connection to recursive identification

In this section, we give an overview of recursive prediction-error methods (RPEM) and discuss a unified view of online parameter estimation methods inspired by RPEM.

3.1. System identification

The goal of system identification is to infer a model of interacting variables and signals from observations. A mapping of datasets of observations to parameter space is referred to as an *identification method* (Ljung, 1998). There are two classes of identification methods: batch (offline), and recursive (online). Batch methods store the entire sequence of observations up to time t and use it to estimate the true (unknown) model parameter θ. In recursive identification, the estimate of θ at time t is computed only from the previous estimate and a fixed-size state. Due to their modest memory requirements, recursive identification methods are well-suited to online parameter estimation in deep learning.

3.2. Recursive prediction-error methods

Define $e_t(\hat{\mathrm{y}})$ to be a non-negative scalar measure of the prediction error $\hat{\mathrm{y}}$ at time t, and $\hat{\theta}$ to be an estimate of the true (unknown) estimation model parameter. *Recursive prediction-error methods* (RPEM) are a family of approaches to recursive identification that aim to choose $\hat{\theta}$ to minimize these errors aggregated up to and including the current time t. This is formalized by the following objective:

$$V_t(\theta) = \sum_{k=0}^{t} \Big(\prod_{i=k+1}^{t} \lambda_i \Big) e_k(\hat{\mathrm{y}}_k) \tag{3}$$

where $\lambda_t \in [0, 1]$, called the forget factor, controls how much we decay past errors at time t. For the linear state estimation model in (2), a common choice for e_t in the recursive identification literature is the quadratic objective: $e_t(\hat{\mathrm{y}}_t) = \frac{1}{2}\varepsilon_t^\top \Lambda^{-1} \varepsilon_t$, where $\varepsilon_t := \mathrm{y}_t^* - \hat{\mathrm{y}}_t$ is called the

[2] A more general version of (2b) would be $\hat{\mathrm{y}}_t = H(\theta)\hat{\mathrm{x}}_t$, with $\mathrm{y}, \hat{\mathrm{y}} \in \mathbb{R}^m$ and $H(\theta) \in \mathbb{R}^{m \times n}$. In this paper, we consider first order models with $H = I$ only, and include (2b) for notational consistency.

innovation, y_t^* is the target value for the prediction $\hat{\mathsf{y}}_t$, and $\Lambda \in \mathbb{R}^{n\times n}$ is positive-definite.

3.3. Derivation of the RPEM update

In this section, we derive the RPEM update for the parameter estimate $\hat{\theta}$ for a linear state estimation model as in (2), and a general error measure e_t.

Define the Jacobian $\psi_t(\theta) = \frac{\partial \hat{\mathsf{y}}_t}{\partial \theta}(\theta)$ and $\psi_t := \psi_t(\hat{\theta}_{t-1})$. Given an estimate $\hat{\theta}_{t-1}$ and observation y_t, we seek to compute a new estimate $\hat{\theta}_t$ using a fixed-size state. To do so, RPEM makes the following assumptions:

1. *Optimality on past observations:* $V'_{t-1}(\hat{\theta}_{t-1}) = 0$, i.e., the previous model estimate $\hat{\theta}_{t-1}$ was optimal for the objective V_{t-1}.
2. *Slow-moving parameters:* To compute the Jacobian ψ_t exactly would incur a memory cost that grows proportional to t, since a prediction $\hat{\mathsf{y}}_t$ depends on θ through the entire history of observations up to time t. Instead, we will compute ψ_t recursively using only ψ_{t-1}. In effect, this is making the assumption that $\psi_{t-1}(\hat{\theta}_{t-2}) \approx \psi_{t-1}(\hat{\theta}_{t-1})$, which holds for slow-moving parameters, i.e., when $\hat{\theta}_{t-1} \approx \hat{\theta}_{t-2}$.

By the first assumption, $V'_t(\hat{\theta}_{t-1}) = \psi_t \nabla_{\hat{\mathsf{y}}} e_t(\hat{\mathsf{y}}_t)$. Applying Newton's method to solve $V'_t(\hat{\theta}_t) = 0$ gives the update:

$$\hat{\theta}_t = \hat{\theta}_{t-1} - V''_t(\hat{\theta}_{t-1})^{-1}\psi_t \nabla_{\hat{\mathsf{y}}} e_t(\hat{\mathsf{y}}_t) \tag{4}$$

Let $R_t = \gamma_t V''_t(\hat{\theta}_{t-1})$ with $\gamma_t = (\sum_{k=0}^{t}(\prod_{i=k+1}^{t}\lambda_i))^{-1}$ as a normalizing constant. Then, the general form of an RPEM update for $\hat{\theta}$ applied to the model (2) is:

$$\hat{\mathsf{y}}_t = \hat{\mathsf{x}}_t \tag{5a}$$

$$\psi_t = \frac{\partial}{\partial \hat{\mathsf{x}}}\left[A(\hat{\theta}_{t-1})\hat{\mathsf{x}}_{t-1}\right]\big|_{\hat{\mathsf{x}}=\hat{\mathsf{x}}_{t-1}}\psi_{t-1} + \frac{\partial}{\partial \theta}\left[A(\theta)\hat{\mathsf{x}}_{t-1} + B(\theta)\mathsf{y}_{t-1}\right]\big|_{\theta=\hat{\theta}_{t-1}} \tag{5b}$$

$$\hat{\theta}_t = \hat{\theta}_{t-1} - \gamma_t R_t^{-1}\psi_t \nabla_{\hat{\mathsf{y}}} e_t(\hat{\mathsf{y}}_t) \tag{5c}$$

$$\hat{\mathsf{x}}_{t+1} = A(\hat{\theta}_t)\hat{\mathsf{x}}_t + B(\hat{\theta}_t)\mathsf{y}_t \tag{5d}$$

A UNIFYING FRAMEWORK

These equations provide a framework for jointly estimating the state x_t and model parameters θ_t of a linear state model in an online fashion. A realization of this framework is fully specified by choices of: the state model $A(\theta)$ and $B(\theta)$, the Jacobian estimate $\psi_t(\theta)$, the prediction error $e_t(\hat{\mathsf{y}})$, the preconditioner R_t (estimate of the Hessian), and the learning rate γ_t. There are a number of options for estimating the preconditioner and Jacobian, which we discuss below.

3.4. Gauss-Newton algorithm with forget factor

For a general objective, setting the preconditioner $R_t = I$ for all t recovers traditional gradient descent on the objective with step size γ_t. Setting R_t to be the exact Hessian recovers Newton's method, which is known to perform better than gradient descent (Ljung, 1998). However, computing the exact Hessian may be infeasible. Assuming quadratic errors $e_t(\hat{\mathsf{y}}_t) = \frac{1}{2}\varepsilon_t^\top \Lambda^{-1}\varepsilon_t$ that are decayed over time as in (3), an alternative is to use the Gauss-Newton algorithm to recursively estimate the preconditioner.

Define $\bar{P}_t := \gamma_t R_t^{-1}$ where γ_t is the normalizing constant for the forget factors, $\gamma_t = (\sum_{k=0}^{t}(\prod_{i=k+1}^{t}\lambda_i))^{-1}$. The Gauss-Newton update with forget factor λ_t corresponds to the following update for $\bar{P}_t$ (see Appendix A for details):

$$\bar{P}_t = \bar{P}_{t-1}(\lambda_t I + \psi_t \Lambda^{-1}\psi_t^\top \bar{P}_{t-1})^{-1}.$$

The forget factor λ controls the trade-off between the ability to maintain stable estimates at steady state, and the ability to fit time-varying θ. A good heuristic for setting λ_t is the Bounded Gain Forget (BGF) method (Slotine et al., 1991):

$$\lambda_t = \lambda_{\max}\left(1 - \frac{\|\bar{P}\|}{k_0}\right).$$

$\lambda_{\max} < 1$ is the maximum forget factor value, and is usually set to be close to 1. k_0 is the upper bound for $\|\bar{P}\|$, and is usually set to a large constant. (See Appendix A for details.)

Using the preconditioner $\bar{P}$ removes the need to choose a learning rate sequence γ_t, instead using a forget factor parameterization. In this case, it can be shown that the effective learning rate γ_t is given by:

$$\gamma_t = \gamma_{t-1}/(\lambda_t + \gamma_{t-1}).$$

Moreover, if θ is a scalar, such as an EMA decay parameter, $\bar{P}_t$ can be interpreted as an adaptive scalar learning rate for the update of $\hat{\theta}$.

3.5. Discounted Jacobian

A variation on the Jacobian computation in (5b) is the discounted Jacobian, where $\mu_t \in [0, 1]$ is a hyperparameter:

$$\psi_t = \mu_t \frac{\partial}{\partial \hat{\mathsf{x}}}\left[A(\hat{\theta}_{t-1})\hat{\mathsf{x}}_{t-1}\right]\big|_{\hat{\mathsf{x}}=\hat{\mathsf{x}}_{t-1}}\psi_{t-1} + \frac{\partial}{\partial \theta}\left[A(\theta)\hat{\mathsf{x}}_{t-1} + B(\theta)\mathsf{y}_{t-1}\right]\big|_{\theta=\hat{\theta}_{t-1}} \tag{6}$$

Choosing $\mu = 0$ corresponds to truncating the computation graph to ignore the dependency that past states had on the model parameter θ. Although this would make the Jacobian computation biased, it is more computationally efficient. On the other hand, choosing $\mu = 1$ leads to more accurate Jacobian approximations near steady-state (when $\hat{\theta}$ does not change much), but may have poor transient performance due to bad initial values of ψ. Ljung & Söderström (1983) suggests a simple trick to improve the transient performance, by setting μ_t to exponentially approach 1:

$$\mu_t = 1 - \nu^t, \quad t = 0, 1, 2, \ldots, \text{typically } \nu \approx 0.99 \tag{7}$$

When a non-zero μ is used, storing the Jacobian ψ_t incurs an additional memory cost proportional to $n \times d$. This makes RPEM particularly well-suited for hyperparameter optimization, where d, the number of hyperparameters being adapted, is typically small.

4. ALiSE

In this section, we introduce ALiSE (Algorithm 1) for jointly estimating statistics $\hat{\mathrm{x}}$ and model parameters θ (such an an EMA decay parameter) in a neural network training context. ALiSE is a direct result of applying the RPEM framework from Section 3 to a linear state estimation model in (2), where:

- An incoming observation y_t is the noisy statistic $\hat{\mathrm{x}}_t^{\mathrm{B}}$ computed on a batch B.
- The prediction error is a quadratic objective $e_t(\hat{\mathrm{y}}_t) = \frac{1}{2}\varepsilon_t^\top \hat{\Lambda}_t^{-1}\varepsilon_t$ where the target value y_t^* in the innovation ε_t is computed using another estimate of the statistic.
- The Jacobian ψ is estimated using a discount factor $\mu_t \in [0, 1]$ as in (6).
- The preconditioner R_t and learning rate γ_t are computed using the Bounded Gain Forget (BGF) method (Section 3.4).

Below we discuss the choices of prediction error and Jacobian estimate used in ALiSE in more detail.

4.1. The objective

ALiSE uses the prediction error function $e_t(\hat{\mathrm{y}}_t) = \frac{1}{2}\varepsilon_t^\top \hat{\Lambda}_t^{-1}\varepsilon_t$ where $\varepsilon_t = \mathrm{y}_t^* - \hat{\mathrm{y}}_t$. The estimate at time t is $\hat{\mathrm{y}}_t = \hat{\mathrm{x}}_t^{\mathrm{B}}$. To determine y_t^*, we use an unbiased estimate of the statistic computed on another batch B$'$, i.e. $\mathrm{y}_t^* = \hat{\mathrm{x}}_t^{B'}$.

Choosing $\hat{\Lambda}_t = I$ leads to minimizing the norm of the innovation. Ljung (1998) shows that, assuming the innovations ε_t are i.i.d with zero mean, choosing $\hat{\Lambda}_t$ to be the covariance matrix of the innovations ε_t is optimal in the sense that the variance of the estimator $\hat{\theta}_t$ is minimized.

Since it is hard to determine the covariance matrix a priori, we can initialize it to the identity and estimate it online (Ljung & Söderström, 1983):

$$\hat{\Lambda}_t = \hat{\Lambda}_{t-1} + \gamma_t(\varepsilon_t\varepsilon_t^\top - \hat{\Lambda}_{t-1}).$$

However, keeping track of a full $n \times n$ matrix can be computationally infeasible in practice. Instead, we use the following approximation for $\hat{\Lambda}$, only storing its diagonal entries:

$$\hat{\Lambda}_t = \hat{\Lambda}_{t-1} + \gamma_t(\mathrm{diag}(\varepsilon_t\varepsilon_t^\top) - \hat{\Lambda}_{t-1}) \tag{8}$$

4.2. Detached Jacobian

It is important to note that ALiSE estimates statistics in a way that is detached from the neural network optimization. In theory, an estimate $\hat{\mathrm{x}}_t$ depends on previous estimates through both the state estimation model, and the neural network weights. In Figure 1, this corresponds to the two different paths that connect $\hat{\mathrm{x}}_{t-1}$ to $\hat{\mathrm{x}}_t$. However, ALiSE ignores the path through the neural network weights w_{t-1}. This is a simplifying assumption that allows us to reduce the first term in the Jacobian (5b) as below:

$$\psi_{t+1} = \mu_t A(\hat{\theta}_{t-1})\psi_t + \frac{\partial}{\partial\theta}\left[A(\theta)\hat{\mathrm{x}}_t + B(\theta)\mathrm{y}_t\right]\big|_{\theta=\hat{\theta}_t}$$

Algorithm 1 ALiSE

```
Given (at time $t$): forget factor $\lambda_t$, learning rate $\gamma_t$, observations $\mathrm{y}_t$, $\mathrm{y}_t^*$, discount $\mu_t$, Jacobian estimate $\psi_t$.
$\varepsilon_t = \mathrm{y}_t^* - \mathrm{y}_t$
$\hat{\Lambda}_t = \hat{\Lambda}_{t-1} + \gamma_t(\mathrm{diag}(\varepsilon_t\varepsilon_t^\top) - \hat{\Lambda}_{t-1})$
$\bar{P}_t = \bar{P}_{t-1}(\lambda_t I + \psi_t\hat{\Lambda}_t^{-1}\psi_t^\top \bar{P}_{t-1})^{-1}$
$\hat{\theta}_t = \hat{\theta}_{t-1} - \bar{P}_t\psi_t\hat{\Lambda}_t^{-1}\varepsilon_t$
$\hat{\mathrm{x}}_{t+1} = A(\hat{\theta}_t)\hat{\mathrm{x}}_t + B(\hat{\theta}_t)\mathrm{y}_t$
$\hat{\mathrm{y}}_{t+1} = \hat{\mathrm{x}}_{t+1}$
$\psi_{t+1} = \mu_t A(\hat{\theta}_t)\psi_t + \frac{\partial}{\partial\theta}\left[A(\theta)\hat{\mathrm{x}}_t + B(\theta)\mathrm{y}_t\right]\big|_{\theta=\hat{\theta}_t}$
return $\hat{\mathrm{y}}_{t+1}$
```

5. Related works

System identification There is a long history of research in online state estimation with unknown model parameters. The field was referred to by Åström & Eykhoff (1971) as "a fiddler's paradise", because the literature had once been diverse and scattered. Examples include the recursive least squares method (Plackett, 1950), variants of the ARMA model (Box et al., 2015) common in time-series forecasting, Bayesian (nonlinear) filtering methods such as the (extended) Kalman Filter (Kalman, 1960), and, in control theory, methods such as Model Reference Adaptive Control (MRAC) (Åström & Wittenmark, 2013).

Ljung & Söderström (1983) states that: "*There is only one recursive identification method. It contains some design variables to be chosen by the user.*" In this spirit, our work presents a unified view of recursive approaches to online parameter estimation in deep learning. ALiSE , our proposed approach for linear state estimation, draws upon a subset of methods known as recursive prediction-error methods

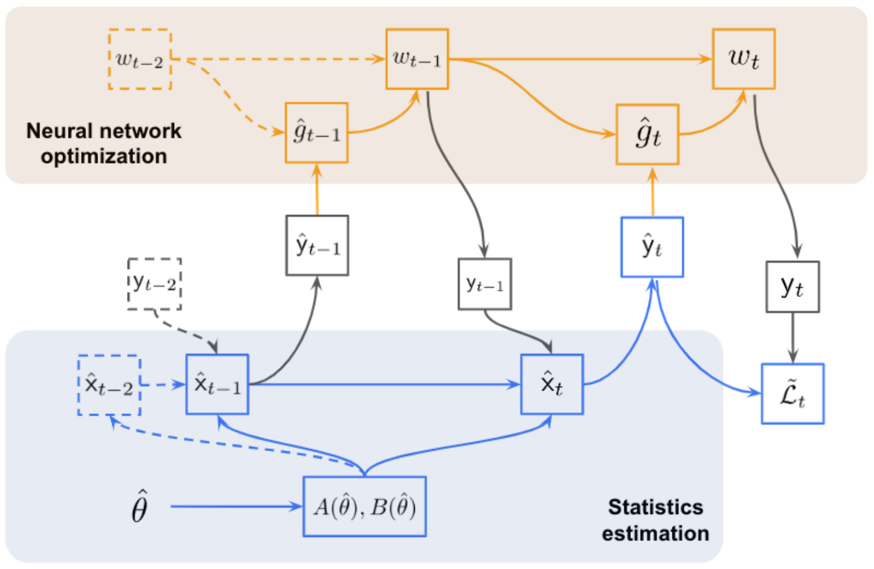


Figure 1. The computation graph of ALiSE applied to neural network optimization. $\tilde{\mathcal{L}}_t$ denotes the network training objective.

(Ljung, 1998), with some modifications to suit deep learning applications. Specifically, we use a diagonal approximation for the quadratic objective weights (8) to reduce the memory and computation cost (see Section 4.1). For learning rate selection (gain scheduling), we use the Bounded Gain Forget (BGF) method from Slotine et al. (1991). For estimating the Jacobian (6), we apply a discounting mechanism similar to Donini et al. (2020). Similar discounting strategies such as TD(λ) (Sutton & Barto, 2018) and generalized advantage estimation (Schulman et al., 2016) have been used in reinforcement learning.

Learning rate scheduling The recursive computation of the Jacobian in RPEM has previously been applied in several machine learning contexts, often referred to as *forward-mode differentiation*. Hypergradient descent (HD) (Baydin et al., 2018), Stochastic Meta-Descent (SMD) (Schraudolph, 1999), MARTHE (Donini et al., 2020), and Sutton (1992) all use forward-mode differentiation to compute the Jacobian of network weights with respect to a learning rate parameter, and use it for online adaptation of the learning rate. Table 1 shows how many of these online parameter estimation methods can be explicitly cast as recursive prediction-error methods with particular model and design choices.

Real-time recurrent learning Forward-mode differentiation is also used in real-time recurrent learning (RTRL), a classical algorithm to train recurrent neural networks (Williams & Zipser, 1989).[3] More recently, Im et al. (2021)

Table 1. RPEM interpretation of various online parameter estimation methods in deep learning

	MARTHE	SMD	HD	RTRL
θ	LR α	LR α	LR α	Weights w
$\hat{x}$	Weights w	Weights w	Weights w	RNN state
y	$g(\cdot)$	$g(\cdot)$	$g(\cdot)$	RNN input
A, B	$I, -\alpha I$	$I, -\alpha I$	$I, -\alpha I$	-
e	Val. loss	Train. loss	Train. loss	Train. loss
ψ	$\mu \in (0,1)$	$\mu = 1$	$\mu = 0$	$\mu = 1$

draw an explicit connection between training an RNN using RTRL, and online hyperparameter optimization (OHO). In their OHO framework, the state being jointly adapted with the hyperparameters are the neural network weights. In contrast, ALiSE is used for statistics estimation and goes beyond OHO by borrowing several design choices from the recursive identification literature.

6. Experiments

6.1. Estimating the second moment in Adam optimizer

The Adam optimizer (Kingma & Ba, 2014) requires estimation of the second moment of the gradient from noisy gradient samples. In particular, the Adam update is scaled by the square root of the exponential moving average (EMA) of squared past gradients. Although the Adam optimizer with its default configuration ($\beta_1 = 0.9, \beta_2 = 0.999$) has been successful in many practical applications, Reddi et al. (2018) point out that the EMA with constant decay parame-

[3] In fact, applying the extended Kalman filter (closely related to RPEM, see discussion in Ljung & Söderström (1983)) jointly on the hidden state and parameter is equivalent to natural gradient on top of RTRL (Ollivier, 2018).

ter can cause non-convergence to the optimal solution even in some convex optimization settings. We investigate two settings where training does not converge using the default configuration, but converges when we use ALiSE to automatically adapt β_2.

6.1.1. Synthetic example

Consider the following simple synthetic example in Reddi et al. (2018):

$$\textbf{Online: } f_t(x) = \begin{cases} 1010x, & t \bmod 101 = 1 \\ -10x, & \text{otherwise} \end{cases}$$

$$\textbf{Stochastic: } f_t(x) = \begin{cases} 1010x, & \text{with probability } \frac{1}{101} \\ -10x, & \text{otherwise} \end{cases}$$

The optimal solution is $x = -1$. However, an Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ converges to a highly sub-optimal solution $x = +1$. This is because the large gradient (1010 in this case) is sparse enough that for the given β_2 value (0.99), it is unable to move x in the correct direction. The same effect happens in both the online and stochastic settings (Figure 2).

We apply ALiSE to this synthetic example, with $\beta_1 = 0.9$ and β_2 initialized to 0.99. As shown in Figure 2, when ALiSE is applied, the optimizer avoids the sub-optimal solution $x = +1$ in both the online and stochastic settings. The β_2 schedules found by ALiSE are shown in Appendix B.1.

6.1.2. Learning word embeddings

Another scenario where the default Adam optimizer may fail to converge is when learning word embeddings. As the gradient signal for learning the embedding for each word is sparse, we require the optimizer to have "long-term memory" of past gradients. We experiment with a simple Word2Vec continuous bag-of-words architecture (Mikolov et al., 2013), trained on the "news" genre of the Brown corpus (Francis & Kucera, 1979). As shown in Figure 3, when using learning rate 0.001, Adam with the default configuration $\beta_1 = 0.9$, $\beta_2 = 0.999$ fails to converge. On the other hand, ALiSE is able to find a schedule for β_2 such that training converges.

6.2. Solving the Schrödinger equation for a 2D hydrogen atom using the Spectral Inference Network

The Spectral Inference Network (SpIN) (Pfau et al., 2018) is a framework for learning eigenfunctions of linear operators by stochastic optimization. Let $u_w(x) \in \mathbb{R}^K$ be the features parameterized by network parameters w, with which we aim to recover the top K eigenfunctions of a symmetric kernel k. Define the (kernel-weighted) feature covariance as $\Sigma := \mathbb{E}_{\mathbf{x}}[u_w(\mathbf{x})u_w(\mathbf{x})^\top]$ and $\Pi := \mathbb{E}_{\mathbf{x},\mathbf{x}'}[k(\mathbf{x},\mathbf{x}')u_w(\mathbf{x})u_w(\mathbf{x}')]$, and let L be the Cholesky decomposition of Σ. SpIN learns *ordered* eigenfunctions by performing gradient ascent, where the gradient [4] is given by:

$$\begin{aligned}\text{Gradient} = \mathbb{E}[J_\Pi(L^{-\top}\mathrm{diag}(L)^{-1})] \\ - \mathbb{E}[J_\Sigma(L^{-\top}\mathrm{triu}(L^{-1}\Pi L^{-\top}\mathrm{diag}(L)^{-1}))],\end{aligned} \tag{9}$$

where triu and diag returns the upper triangular and diagonal of a matrix, respectively. $J_\Pi(\cdot)$, $J_\Sigma(\cdot)$ are linear operators that denote left-multiplication of the Jacobian of Π and Σ with respect to w.

Note that (9) is a nonlinear function of Σ and J_Σ, which are themselves expectations. During training, we only have access to empirical estimates $\hat{\Sigma}$ and $\hat{J}_\Sigma$ from samples. Naively plugging them into (9) will result in a biased gradient. Pfau et al. (2018) addressed this issue by using the exponential averaged statistics $\bar{\Sigma}_t$ and $\bar{J}_{\Sigma_t}$ in (9) instead of Σ and J_Σ:

$$\begin{bmatrix}\bar{\Sigma}_t \\ \bar{J}_{\Sigma_t}\end{bmatrix} = \beta \begin{bmatrix}\bar{\Sigma}_{t-1} \\ \bar{J}_{\Sigma_{t-1}}\end{bmatrix} + (1-\beta)\begin{bmatrix}\hat{\Sigma}_t \\ \hat{J}_{\Sigma_t}\end{bmatrix}. \tag{10}$$

The performance of SpIN is quite sensitive to the value of β, which is hand-set by the authors to a constant value. We show that by replacing the EMA with ALiSE, we can accelerate the convergence of SpIN without the need to manually tune β. We consider an experiment in Pfau et al. (2018), where SpIN is used to solve the Schrödinger equation of a 2-dimensional hydrogen atom. In particular, we use SpIN to train a neural network to approximate the wavefunctions $\Psi(\mathbf{x})$, which are solutions of the time-independent Schrödinger equation:

$$E\Psi(\mathbf{x}) = \frac{-\hbar}{2m}\nabla^2\Psi(\mathbf{x}) + V(\mathbf{x})\Psi(\mathbf{x}). \tag{11}$$

We use the same setup in Pfau et al. (2018), including setting $\frac{\hbar}{2m}$ to 1 and choosing the potential $V(\mathbf{x}) = \frac{1}{|\mathbf{x}|}$. The Schrödinger equation for the 2D hydrogen atom can be solved analytically (Yang et al., 1991), allowing us to evaluate the neural network approximations.

We define the following metric d_{align}:

$$\frac{1}{K}\sum_{i=1}^{K}\Big(1 - \frac{\hat{\Psi}_{w,i}^\top \Phi_{E_{n(i)}}(\Phi_{E_{n(i)}}^\top \Phi_{E_{n(i)}})^{-1}\Phi_{E_{n(i)}}^\top \hat{\Psi}_{w,i}}{\|\hat{\Psi}_{w,i}\|_2^2}\Big) \tag{12}$$

d_{align} measures how aligned the learned eigenfunctions are to the analytical solutions (the definition is similar to the axis-alignment distance in Bao et al. 2020). $\hat{\Psi}_{w,i}$ is the learned approximation of the i^{th} eigenfunction. $n(i)$ is the principal quantum number (which determines the energy level E_n) corresponding to the eigenfunction index i. $\Phi_{E_{n(i)}} := \begin{bmatrix}\Psi_{E_{n(i)},1} & \ldots \Psi_{E_{n(i)},2n-1}\end{bmatrix}$ is a concatenation

[4]The "gradient" is modified and not the actual gradient of a loss function. See Pfau et al. (2018) for details.

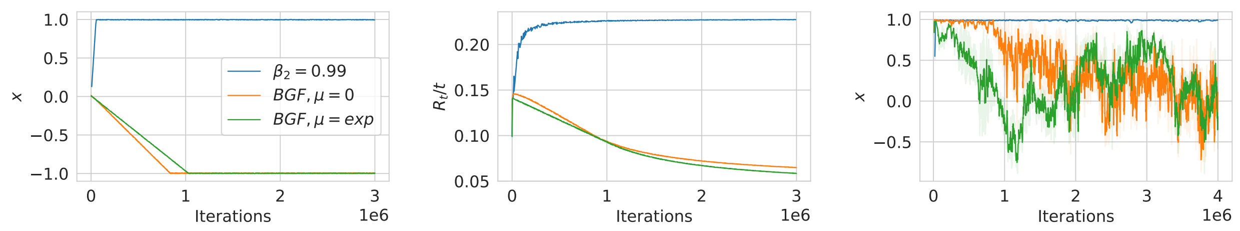
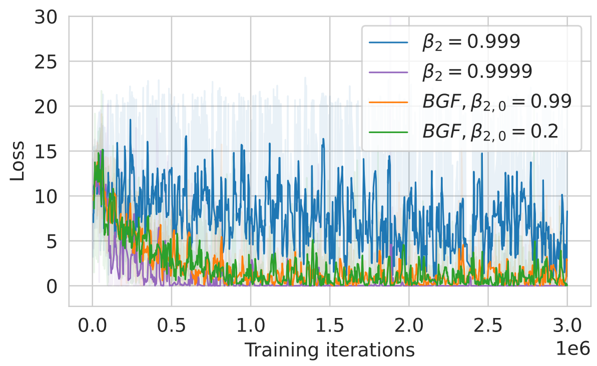


Figure 2. A synthetic example where the Adam baseline does not converge to the optimal solution. Left and centre: online setting. Right: stochastic setting. The configuration $\mu = \exp$ refers to the exponentially decaying schedule in (7)

.



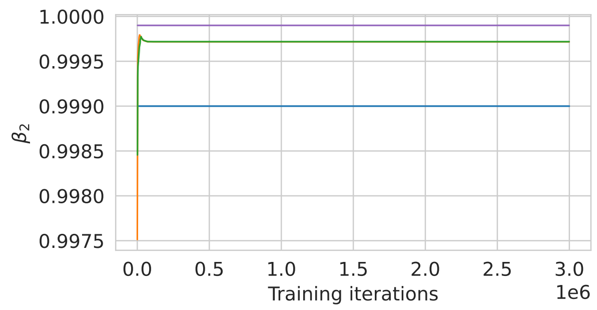


Figure 3. Word2Vec continuous bag-of-words trained on the "news" genre of the Brown corpus.

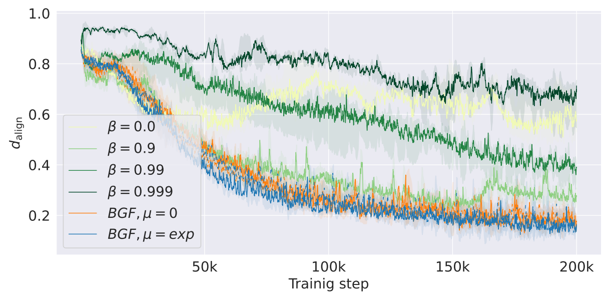


Figure 4. Learning the first 9 eigenfunctions of the Schrödinger equation of a 2D hydrogen atom using SpIN. For experiments that use ALiSE, labeled as BGF (Bounded Gain Forget), solid lines correspond to constrained-sum cases ($A(\hat{\theta}) + B(\hat{\theta}) = 1$), and dashed lines correspond to the unconstrained cases.

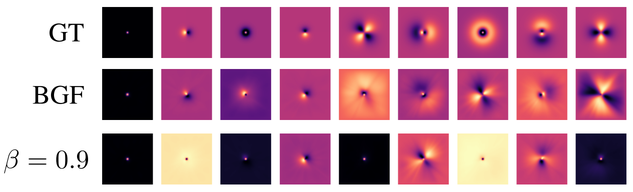


Figure 5. Visualization of the ground truth and learned eigenfunctions.

of the analytical solutions at energy level $n(i)$. Intuitively, d_{align} is the average of the squared sine angles between the first K learned wavefunctions and the vector spaces spanning all the analytical solutions at the same energy level. All the wavefunctions in (12) are evaluated on a 128×128 grid (same as in Pfau et al. (2018)).

We experiment with 4 different configurations of ALiSE that are combinations of: constrained sum / unconstrained sum, and $\mu = 0$ / $\mu = \exp$, where exp refers to the exponentially decaying schedule in (7). In the constrained sum configuration, a single decay parameter β is used. For unconstrained sum, we adapt two decay coefficients jointly and do not constrain them to sum to 1. We also run 4 EMA baselines with $\beta \in \{0, 0.9, 0.99, 0.999\}$. As shown in Figure 4, ALiSE results in faster learning of the eigenfunctions than the baselines, regardless of configuration. Figure 5 visualizes the ground truth and the some of the learned eigenfunctions of the 2D hydrogen atom.

6.3. Learning fair representations

As a last experiment, we consider a classic algorithm for fairness in machine learning: Learning Fair Representations (LFR) (Zemel et al., 2013). LFR achieves fair classifications by mapping each data point in the input space to a new representation space. The new representation obfuscates the sensitive information of the input data, while retaining as much other information as possible to enable good classification performance.

Let S denote the binary random variable that contains the sensitive information. Let $x \in \mathbb{R}^D$ be a feature vector that represents an individual, and $\{X_T, y_T\}$ denote the training set that contains inputs and binary labels. Let $X_T^+, X_T^- \subset X_T$ be the subset of training data that correspond to $S = 1$ and $S = 0$ respectively. Let $m \in \mathbb{R}^K$ be the latent representation for input x, and $v_k \in \mathbb{R}^D$ be prototype (linear decoder) for the k^{th} dimension.

The LFR loss is a weighted sum of three terms: the cross-entropy classification loss $\mathcal{L}_{\mathrm{CE}}$, the reconstruction loss from the intermediate representations $\mathcal{L}_{\mathrm{recon}} = \sum_{x^{(i)} \in X_T} \|x^{(i)} - \sum_{k=1}^K m_k^{(i)} v_k\|_2^2$, and a regularization term $\mathcal{L}_{\mathrm{R}}$ (to encourage statistical parity — a fairness metric) defined as:

$$\mathcal{L}_{\mathrm{R}} = \sum_{k=1}^{K} \left| \frac{1}{|X_T^+|} \sum_{\substack{i=1 \\ x^{(i)} \in X_T^+}}^{N} m_k^{(i)} - \frac{1}{|X_T^-|} \sum_{\substack{i=1 \\ x^{(i)} \in X_T^-}}^{N} m_k^{(i)} \right|. \tag{13}$$

Note that when training with minibatches, we do not have access to the global statistics (highlighted in blue) in (13), and have to resort to online estimates. Also, as the regularization is ℓ_1, a noisy estimate of the statistics would result in biased gradient.

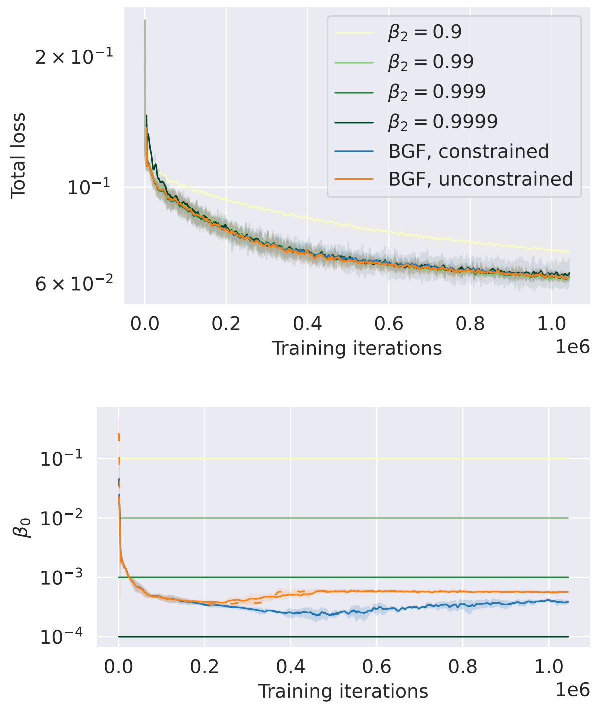


Figure 6. Training loss (evaluated on the whole training set) and the β_0 schedule of LFR on the Adult dataset.

We conduct experiments on the Adult dataset from the UCI repository (Dua & Graff, 2017). The target is to predict whether an individual has income over \$50K. The sensitive feature is *gender*. We fix the weight coefficients for the loss terms $\mathcal{L}_{\mathrm{CE}}$, $\mathcal{L}_{\mathrm{recon}}$ and $\mathcal{L}_{\mathrm{R}}$ to be 1.0, 0.01 and 0.1 respectively. We train LFR end-to-end, with an additional normalization for the prototypes to accelerate training. We use a batch size of 1 to demonstrate the necessity of online statistics estimation. We compare ALiSE (BGF, constrained / unconstrained sum) to naive EMA baselines (decay parameter 0.9, 0.99, 0.999 and 0.9999). As shown in Figure 6, regardless of $\hat{\theta}$ initialization, ALiSE is able to find a parameter schedule that matches the performance of the best naive EMA baselines.

7. Conclusion

Estimating statistics from noisy observations when training a neural network can lead to slower convergence. In this work, we propose ALiSE, an approach to online statistics estimation for neural network training that draws on techniques from recursive identification. Our work establishes an explicit connection between recursive prediction-error methods and online parameter estimation in deep learning. We believe this connection is useful as it allows us to borrow design choices from the recursive identification literature, as ALiSE does. Moreover, when the number of parameters being adapted is small, the additional memory requirement of these recursive methods is modest, making them well-suited for hyperparameter optimization in deep learning. We show that ALiSE is able to learn sensible schedules for an EMA decay parameter in many applications, leading to matching or accelerated convergence during training. We hope this work will inspire further applications of recursive identification in deep learning.

References

Åström, K. J. and Eykhoff, P. System identification—a survey. *Automatica*, 7(2):123–162, 1971.

Åström, K. J. and Wittenmark, B. *Adaptive control*. Courier Corporation, 2013.

Bao, X., Lucas, J., Sachdeva, S., and Grosse, R. B. Regularized linear autoencoders recover the principal components, eventually. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6971–6981, 2020.

Baydin, A. G., Cornish, R., Martínez Rubio, D., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M.

Time series analysis: forecasting and control. John Wiley & Sons, 2015.

Donini, M., Franceschi, L., Majumder, O., Pontil, M., and Frasconi, P. Marthe: Scheduling the learning rate via online hypergradients. In *International Joint Converence on Artificial Intelligence*, 2020.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Francis, W. N. and Kucera, H. Brown corpus manual. *Letters to the Editor*, 5(2):7, 1979.

Im, D. J., Savin, C., and Cho, K. Online hyperparameter optimization by real-time recurrent learning. *arXiv preprint arXiv:2102.07813*, 2021.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ljung, L. *System Identification: Theory for the User, 2nd Edition*. Pearson Education, 1998.

Ljung, L. and Söderström, T. *Theory and practice of recursive identification*. MIT press, 1983.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Ollivier, Y. Online natural gradient as a kalman filter. *Electronic Journal of Statistics*, 12(2):2930–2961, 2018.

Pfau, D., Petersen, S., Agarwal, A., Barrett, D. G., and Stachenfeld, K. L. Spectral inference networks: Unifying deep and spectral learning. In *International Conference on Learning Representations*, 2018.

Plackett, R. L. Some theorems in least squares. *Biometrika*, 37(1/2):149–157, 1950.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

Schraudolph, N. N. Local gain adaptation in stochastic gradient descent. In *Proceedings of the 9th International Conference on Neural Networks (ICANN)*, volume 2, pp. 569–574, 1999.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

Slotine, J.-J. E., Li, W., et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

Sutton, R. S. Gain adaptation beats least squares? In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 161–166, 1992.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Yang, X., Guo, S., Chan, F., Wong, K., and Ching, W. Analytic solution of a two-dimensional hydrogen atom. i. nonrelativistic theory. *Physical Review A*, 43(3):1186, 1991.

Zemel, R., Wu, Y., Swersky, K., Pitassi, T., and Dwork, C. Learning fair representations. In *International conference on machine learning*, pp. 325–333. PMLR, 2013.

A. Details of the Gauss-Newton algorithm with forget factor

We use a recursive approximation (denoted as R) of the Gauss-Newton Hessian $\mathbb{E}[\psi\Lambda^{-1}\psi^\top]$:

$$R_t = R_{t-1} + \gamma_t(\psi_t\hat{\Lambda}_t^{-1}\psi_t^\top - R_{t-1}). \tag{14}$$

The preconditioning matrix is $P_t = R_t^{-1}$. Using the forget factor parameterization and defining $\bar{P} := \gamma P = \gamma R^{-1}$, it can be shown that (14) corresponds to the following recursive update for $\bar{P}$:

$$\bar{P}_t = \big(\bar{P}_{t-1} - \bar{P}_{t-1}\psi_t(\psi_t^\top\bar{P}_{t-1}\psi_t + \lambda_t\hat{\Lambda}_t)^{-1}\psi_t^\top\bar{P}_{t-1}\big)/\lambda_t = \bar{P}_{t-1}(\lambda_t I + \psi_t\hat{\Lambda}_t^{-1}\psi_t^\top\bar{P}_{t-1})^{-1} \tag{15}$$

The forget factor λ controls the trade-off between the steady-state state estimation performance and the ability to fit time-varying θ. If $\forall t, \lambda_t = 1$, the algorithm never "forgets" past observations, $\bar{P}$ will approach 0 over time, and $\hat{\theta}$ will eventually stop changing. This is undesirable if the optimal θ is time-varying. On the other hand, having a fixed $\lambda_t = \lambda < 1$ will cause $\|\bar{P}\| \to \infty$ if the second term inside the parenthesis in (15) is sufficiently small. In order to fit a time-varying θ without causing instability, we use the Bounded Gain Forget (as it keeps $\|\bar{P}\|$ bounded (Slotine et al., 1991)) for choosing the forget factor:

$$\lambda_t = \lambda_{\max}(1 - \frac{\|\bar{P}\|}{k_0}).$$

B. Additional experiment details

B.1. Synthetic example in estimating the second moment in Adam optimizer

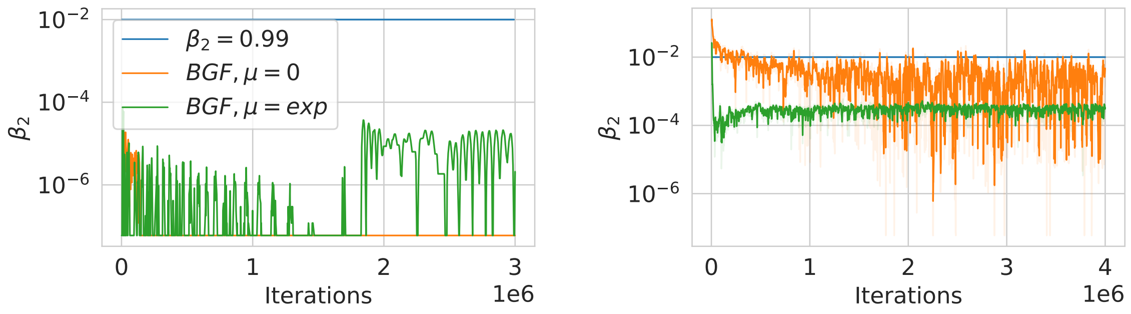


Figure 7. The β_2 schedules found by ALiSE (BGF), when applied on the synthetic example where the baseline Adam optimizer does not converge to the optimal solution. Left: online setting. Right: stochastic setting.