A Robust Control Framework for Stochastic Neural Network Training

Xuchan Bao *jennybao@cs.toronto.edu*
Department of Computer Science
University of Toronto, Vector Institute

John W. Simpson-Porco *jwsimpson@ece.utoronto.ca*
Department of Electrical and Computer Engineering
University of Toronto

Roger Grosse *rgrosse@cs.toronto.edu*
Department of Computer Science
University of Toronto, Vector Institute

1 Introduction

Deep learning models have achieved remarkable success across numerous domains, from computer vision (Krizhevsky et al., 2012; He et al., 2016) to natural language processing (Devlin et al., 2019; Brown et al., 2020), as well as protein structure prediction (Jumper et al., 2021) and game playing (Silver et al., 2017). Despite these achievements, efficiently training these models remains challenging and poorly understood. Even with automated hyperparameter search and extensive computational resources, practitioners still rely on empirical wisdom and heuristics to select optimizers, learning rates, batch sizes, momentum coefficients, and other crucial hyperparameters. We lack a comprehensive theoretical understanding of how these various factors interact to affect training dynamics.

One fruitful approach to understanding neural network optimization is to develop simple toy models that can provide insights into real training dynamics. Ideally, such models should be simple enough to allow for cheap experimentation and theoretical analysis, while capturing essential aspects of neural network training so that insights transfer to realistic settings. A popular example of such a toy model is the Noisy Quadratic Model (NQM) (Zhang et al., 2019). The NQM assumes a quadratic loss function with additive gradient noise, allowing for closed-form analysis of optimization trajectories. By running simple simulations of the NQM, researchers have derived important insights about how factors such as momentum, moving averages, and preconditioning affect the benefits of scaling up batch sizes, particularly in understanding the transition between noise-dominated and curvature-dominated optimization regimes.

While the NQM has proven valuable, it has two prominent limitations that prevent it from fully capturing the dynamics of modern neural network training. First, the NQM models minibatch noise as additive with covariance $\Sigma/\mathcal{B}$ (where Σ is the gradient noise covariance and $\mathcal{B}$ is the batch size), but empirical studies have shown that minibatch noise is fundamentally multiplicative — its scale depends on the current loss value (Ziyin et al.; Mori et al., 2022). This distinction is crucial for understanding optimization dynamics throughout training. Second, the NQM assumes entirely linear training dynamics due to its quadratic loss assumption. Although modern neural network training exhibits surprisingly linear dynamics (Jacot et al., 2018; Lee et al., 2019), especially early in the training process, nonlinearities are still necessary for feature learning and state-of-the-art performance (Chizat et al., 2019; Yang & Hu, 2021; Vyas et al., 2022; Fort et al., 2020). The NQM is thus inadequate for modeling the nonlinearities present in neural network training.

In this work, we adopt a control-theoretic framework that addresses these limitations. Following Lessard et al. (2016), we view optimizers as dynamical systems and analyze their convergence using robust control theory. Our contributions extend this framework in two ways: 1) we model minibatch noise as multiplicative, where noise scales with the current loss value; 2) we allow perturbations to the network Jacobian (partial derivatives of outputs with respect to parameters), which introduces nonlinearity into the optimization dynamics.

Another key advantage of our framework is its flexibility. Unlike the NQM, which requires closed-form solutions and laborious derivations for each optimizer, our approach works for any optimizer expressible as a linear time-invariant system. The NQM analysis in Zhang et al. (2019) focuses on heavy-ball momentum, and extending it to other optimizers like Nesterov accelerated gradient (Nesterov, 1983) would require deriving new analytical solutions. In contrast, we can seamlessly handle such optimizers. In fact, our framework reveals important differences between heavy-ball and NAG (Section 4.1) that align with practical observations.

Finally, we note that the scope of this chapter is to analyze the asymptotic performance of optimization algorithms using our framework. We focus on asymptotic convergence properties to provide insights into the long-term behavior and stability characteristics of different optimizers. While practical neural network training often involves early-stopping and exhibits rapid decrease in training loss at early stages, analyzing finite-step performance is beyond the scope of this chapter. Future research could explore finite-step performance analysis, potentially following the line of work on Performance Estimation Problem (PEP) (Drori & Teboulle, 2014; Taylor et al., 2017; 2018; Taylor & Bach, 2019).

2 Background

We introduce the background necessary for our framework in Section 3. We introduce the problem setup in Section 2.1, discuss the Noisy Quadratic Model (NQM) in Section 2.2, and express the NQM in a state-space representation in Section 2.3. A table of notations is provided in Appendix A.

2.1 Problem setup

Consider a machine learning model $f_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by $\theta \in \mathbb{R}^d$, where $\mathcal{X} \subseteq \mathbb{R}^m$ is the input space and $\mathcal{Y} \subseteq \mathbb{R}$ is the output space. The positive integers d and m are the dimensions of the parameter space and input space, respectively. For simplicity, we focus on single-output regression models. However, our analysis can be easily extended to multi-output models, as discussed in Appendix B.

We write $y = f_\theta(x)$ as the output of the model given the input x. We use the squared-error loss function

$$\ell(y, t) = \frac{1}{2}(y - t)^2,$$

where t denotes the target label associated with input x. Given a training dataset of N samples $(x^{(i)}, t^{(i)})$ for $i \in \{1, \dots, N\}$, let $\boldsymbol{x} = (x^{(1)}, \dots, x^{(N)})$ be the vectorized input, with $\boldsymbol{y}$ and $\boldsymbol{t}$ defined similarly. Also, we denote the model that maps the vectorized input to vectorized output as $\boldsymbol{f}_\theta$, where $\boldsymbol{y} = \boldsymbol{f}_\theta(\boldsymbol{x})$. The *empirical risk* is defined as the mean squared error

$$L(\boldsymbol{y}, \boldsymbol{t}) = \frac{1}{2N} \sum_{i=1}^{N} \ell(y^{(i)}, t^{(i)}),$$

and we let

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{t}; \theta) = \frac{1}{2N} \sum_{i=1}^{N} \ell(f_\theta(x^{(i)}), t^{(i)}) \tag{1}$$

denote the loss as an explicit function of θ. We denote the gradient of $\mathcal{L}$ with respect to θ as

$$g := \nabla_\theta \mathcal{L}(\boldsymbol{x}, \boldsymbol{t}; \theta). \tag{2}$$

The gradient in eq. (2) is also called the full-batch gradient, as it is computed using the whole training set $(\boldsymbol{x}, \boldsymbol{t})$. In full-batch training, we iteratively update the parameters θ using the full-batch gradient signal g, in order to minimize the empirical risk $\mathcal{L}$. We use subscript k to denote the quantities at iterative step k.

Minibatch training In practice, we rarely use the full-batch gradient (eq. (2)) for optimization. Instead, it is common to use minibatch training, where we sample a subset (called minibatch) of the training data at each optimization step k, compute the gradient on the minibatch, and update the parameters using the minibatch gradient. Let $\boldsymbol{x}_k^{\mathcal{B}} = (x^{(i_{1,k})}, x^{(i_{2,k})}, \ldots, x^{(i_{\mathcal{B},k})})$ and $\boldsymbol{t}_k^{\mathcal{B}} = (t^{(i_{1,k})}, t^{(i_{2,k})}, \ldots, t^{(i_{\mathcal{B},k})})$ be the vectorized input and target data for a minibatch at step k respectively, where $\mathcal{B}$ ($1 \leq \mathcal{B} \leq N$) is the minibatch size, and $\{i_{1,k}, i_{2,k}, \ldots, i_{\mathcal{B},k}\}$ are samples from $\{1, \ldots, N\}$ without replacement. [1] The average loss on the minibatch is

$$\mathcal{L}_k^{\mathcal{B}}(\boldsymbol{x}_k^{\mathcal{B}}, \boldsymbol{t}_k^{\mathcal{B}}; \theta) = \frac{1}{\mathcal{B}} \sum_{j=1}^{\mathcal{B}} \ell(f_\theta(x^{(i_{j,k})}), t^{(i_{j,k})}),$$

and we denote the minibatch gradient as

$$g_k^{\mathcal{B}} = \nabla_\theta \mathcal{L}_k^{\mathcal{B}}(\boldsymbol{x}_k^{\mathcal{B}}, \boldsymbol{t}_k^{\mathcal{B}}; \theta).$$

In minibatch training, we would replace g_k in eq. (5a) with $g_k^{\mathcal{B}}$. Minibatch training is often preferred to full-batch training for efficiency reasons, as a small subset of the training data can usually provide a good approximation for gradient computation, while significantly reducing compute for each optimization step. In addition, minibatch training has implicit regularization effect, which leads to generalization benefits compared to full-batch training (Keskar et al., 2017; Smith et al., 2021).

2.2 Noisy quadratic model (NQM)

Zhang et al. (2019) propose to analyze the training dynamics of a neural network using a simple *noisy quadratic model* (NQM), which gives remarkably consistent predictions on the effect of batch size in real neural network training. In the NQM model, the loss function (1) is approximated as a quadratic function of the parameters θ as

$$\mathcal{L}_{\text{NQM}}(\theta) = \frac{1}{2} \theta^\top H \theta,$$

where $H \succ 0$ is the Hessian matrix of the NQM loss function with respect to θ. Furthermore, each gradient query is assumed to be corrupted with additive noise ω,

$$g_{\text{NQM}}(\theta) = H\theta + \omega, \tag{3}$$

where $\mathbb{E}[\omega] = 0$ and $\text{Cov}[\omega] = \Sigma \succeq 0$. Importantly, to proceed with their analysis, Zhang et al. (2019) make the assumption that H and Σ are co-diagonalizable (in their simulations, they assumed $H = \Sigma$). This assumption is theoretically motivated by the observation that in an online setting, when the model is well-trained, the Hessian closely matches the gradient covariance (Martens, 2020), and is also supported by empirical evidence (Zhang et al., 2019).

Zhang et al. (2019) assumes that each gradient query (eq. (3)) is independent, so that a batched gradient query with size $\mathcal{B}$ has inversely-scaled covariance,

$$g_{\text{NQM}}^{\mathcal{B}} = H\theta + \omega^{\mathcal{B}}, \quad \text{where } \mathbb{E}[\omega^{\mathcal{B}}] = 0, \ \text{Cov}(\omega^{\mathcal{B}}) = \Sigma/\mathcal{B}. \tag{4}$$

Using the minibatch gradient model in eq. (4), Zhang et al. (2019) analytically solves the training dynamics of gradient descent with heavy-ball momentum for different batch sizes, and demonstrates that this analytical solution produces accurate predictions on how neural network training dynamics is affected by the batch size in practice.

[1] Ideally, we would like the minibatch at each time step k to be independently sampled, in order to ensure randomness during training. However, this would be inefficient for very large datasets, and in practice it is usually sufficient to shuffle the dataset once, then iterate through the dataset to obtain the minibatches in a round robin fashion (Goodfellow et al., 2016).

Inaccuracy of the NQM noise model for minibatch training While the gradient model in eq. (4) satisfies that the noise magnitude decreases with the batch size, it fails to capture that the magnitude of minibatch gradient noise should be a function of the model parameter θ. For example, in the case where the model f_θ perfectly fits the training data $\{\boldsymbol{x}, \boldsymbol{t}\}$, the gradient should be zero regardless of the choice of minibatch data. However, the NQM noise model in eq. (4) has a constant noise covariance throughout training. Theoretically, it has been shown that minibatch noise contains a term that is proportional to the training loss (Ziyin et al., 2022). In realistic deep neural network settings, it has also been empirically demonstrated that minibatch noise is indeed proportional to the training loss (Mori et al., 2022).

Instead of a model for minibatch gradient, eq. (4) is actually a gradient model in the presence of label noise. Label noise is another common type of gradient noise and is caused by the labels $\boldsymbol{t}$ being noisy or corrupted. Label noise is independent of the model parameters θ, and when assumed to be uncorrelated among samples, it fits the model in eq. (4).

In this paper, we will build upon the NQM model. We keep the label noise as modelled in the NQM (eq. (4)), but propose an additional noise term that more accurately models the minibatch gradient noise (section 3.1).

2.3 State-space representation of NQM

To prepare for our analysis that builds upon the NQM model, we need to first convert the NQM into a state-space representation. We start by introducing state-space models for first-order optimizers, and then move on to rendering the NQM into a state-space representation. Finally, using the state-space representation, we show how to certify the convergence rate of the NQM using robust control tools.

First-order optimizers as state-space models First-order optimizers can be expressed as *state-space models* of the form

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}g_k, \tag{5a}$$

$$\theta_k = \bar{C}\xi_k, \tag{5b}$$

where $\xi \in \mathbb{R}^n$ denotes the internal optimizer state, g_k is the gradient with respect to θ at step k, and $\bar{A}, \bar{B}, \bar{C}$ are fixed-value matrices that characterize the optimizer. The dimension n of the state may vary with different optimizers. As two common examples, for heavy-ball momentum (HB) and Nesterov's accelerated gradient (NAG), we use $n = 2d$ and the matrices $\bar{A}, \bar{B}, \bar{C}$ are given by

$$\textbf{(Example: HB)} \quad \bar{A} = \begin{bmatrix} (1+\beta)I_d & -\beta I_d \\ I_d & 0 \end{bmatrix}, \ \bar{B} = \begin{bmatrix} -\eta I_d \\ 0 \end{bmatrix}, \ \bar{C} = \begin{bmatrix} I_d & 0 \end{bmatrix} \tag{6a}$$

$$\textbf{(Example: NAG)} \quad \bar{A} = \begin{bmatrix} (1+\beta)I_d & -\beta I_d \\ I_d & 0 \end{bmatrix}, \ \bar{B} = \begin{bmatrix} -\eta I_d \\ 0 \end{bmatrix}, \ \bar{C} = \begin{bmatrix} (1+\beta)I_d & -\beta I_d \end{bmatrix}, \tag{6b}$$

where β denotes the momentum parameter and η denotes the learning rate, and I_d is the $d \times d$ identity matrix. Despite the similarities of eqs. (6a) and (6b), HB and NAG have very different dynamics and intuitive interpretations. For a detailed discussion on these two momentum methods, please refer to Section 5.

Modeling the NQM using state-space representation To express the training of an NQM in state-space, we combine eq. (5) with the NQM, by substituting g_k with $g_{\mathrm{NQM},k} = H\theta_k + \omega_k^{\mathcal{B}}$, where $\omega_k^{\mathcal{B}} \sim \mathcal{N}(0, \Sigma/\mathcal{B})$,

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}H\theta_k + \bar{B}\omega_k^{\mathcal{B}} \tag{7a}$$

$$\theta_k = \bar{C}\xi_k. \tag{7b}$$

Equation (7) represents the training dynamics of an NQM in state-space. However, it is difficult to directly work with this form in our analysis, as there are two problem-dependent quantities — the Hessian H and the noise covariance Σ. Instead, we will find it convenient to express the NQM noise $\omega_k^{\mathcal{B}}$ as a transformation of a unit Gaussian noise signal. To do that, we will apply two approximations. First, we make the same assumption as in Zhang et al. (2019) about the noise covariance Σ.

Approximation 1 (NQM noise covariance, as in Zhang et al. (2019))**.** $\Sigma = H$.

Second, we apply the Gauss-Newton approximation for Hessian to express it using the model Jacobian. For a model $f_\theta(\boldsymbol{x})$, we define the Jacobian of the model at step k as the matrix of partial derivatives of the model output with respect to the parameters, evaluated at θ_k

$$J_k := \frac{\partial \boldsymbol{f}_\theta(\boldsymbol{x})}{\partial \theta}\Big|_{\theta=\theta_k} \in \mathbb{R}^{N\times d}.$$

For an NQM, the Jacobian J_k is constant with respect to k, which we simply denote as J. We apply the Gauss-Newton approximation for Hessian, and express the Hessian using the Jacobian.

Approximation 2 (Gauss-Newton approximation for NQM Hessian)**.** $H = J^\top J$.

In approximation 2, we have used the fact that $\nabla^2_{\boldsymbol{yy}} L(\boldsymbol{y}_k, \boldsymbol{t}) = I$, since L is the mean squared error loss. The Gauss-Newton Hessian closely approximates the Hessian H (Nocedal & Wright, 2006, Chapter 10.3).

We can now express the NQM gradient noise $\omega_k^{\mathcal{B}}$ as a transformation of a spherical Gaussian noise signal. Let $\epsilon_k \sim \mathcal{N}(0, \frac{1}{\mathcal{B}} I)$ be a N-dimensional spherical Gaussian noise signal, we have

$$\omega_k^{\mathcal{B}} = J^\top \epsilon_k. \tag{8}$$

Substituting eq. (8) and eq. (7b) into eq. (7a) and applying $H = J^\top J$, we have the state-space representation for NQM.

$$\xi_{k+1} = \underbrace{(\bar{A} + \bar{B} J^\top J \bar{C})}_{A} \xi_k + \underbrace{\bar{B} J^\top}_{B_w} \epsilon_k \tag{9a}$$

$$\theta_k = \bar{C} \xi_k. \tag{9b}$$

Certifying the convergence rate of NQM Once expressed in the state-space, we can apply the robust control framework developed in Lessard et al. (2016) to certify the convergence rate of the NQM.

The robust control framework converts the convergence of an optimization problem into the feasibility of a semi-definite programming (SDP) problem. For an NQM, this is shown in Theorem 1 (proof in Appendix D).

Theorem 1 (NQM asymptotic convergence rate)**.** *Consider the optimization on a noisy quadratic model characterized by* (A, B_w) *(as defined in eq.* (9a)*). If for some* $0 < \rho < 1$*, there exists a* $P \succ 0$ *such that:*

$$A^\top P A - \rho^2 P \prec 0 \tag{10}$$

Then the expected NQM loss $\mathbb{E}[\frac{1}{2}\theta_k^\top H \theta_k]$ *asymptotically converges to a steady-state risk with rate* ρ^2*.*

In practice, the SDP problem can be solved using a convex optimization solver. If the solver finds a feasible solution, then we have guaranteed that the NQM converges with the given rate ρ. The framework, initially developed by Lessard et al. (2016), offers a practical procedure for certifying the convergence rate in optimization.

3 Framework for certifying convergence under minibatch and nonlinearity

In this section, we build upon the NQM model in section 2.3 to develop our framework. Our framework is distinguished from the NQM model in two ways: 1) in Section 3.1, we derive a more accurate model for minibatch gradient noise; 2) Sections 3.2 and 3.3 discuss dimensionality reduction techniques, in order to make our framework computationally tractable; 3) Section 3.4 applies the dimensionality reduction techniques to certify convergence of NQM with minibatch noise; 4) in section 3.5, we add an uncertainty on the Jacobian J to account for nonlinear training dynamics.

3.1 A model for minibatch noise

To derive our model for minibatch noise, we first rewrite the NQM model in eq. (9) in the output space of the model, instead of the parameter space. In particular, we will express eq. (9) in terms of the model output $\boldsymbol{y}_k$ and the output-space gradient $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. In NQM, we have $\boldsymbol{y}_k = J\theta_k$ and $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) = \boldsymbol{y}_k$. We substituting these into eq. (9), by first changing the output signal from θ_k to $\boldsymbol{y}_k$ in eq. (9b), then replacing the $J\bar{C}\xi_k$ term in eq. (9a) with $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. We have

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}J^\top \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) + \bar{B}J^\top \epsilon_k \tag{11a}$$

$$\boldsymbol{y}_k = J\bar{C}\xi_k. \tag{11b}$$

In eq. (11), the state-space representation is written in terms of output-space signals, including the output-space gradient $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$ and the model output $\boldsymbol{y}_k$.

To derive the minibatch noise model, we model the minibatch noise as i.i.d. Bernoulli random variables multiplied to the training data points at every optimization step. Although in practical minibatch training, data points contained in a minibatch are sampled without replacement (the dataset is shuffled at the beginning of the epoch, and the minibatches are taken sequentially from the shuffled dataset), and that different batches might not be truly independent, our approximation with i.i.d. Bernoulli random variables captures the crucial feature of minibatch noise — that it is multiplicative, rather than additive. Let $\mathcal{B}$ be the batch size, the output-space gradient at step k with minibatch size $\mathcal{B}$ can be approximated as:

$$\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) := \frac{N}{\mathcal{B}} \begin{bmatrix} b_k^{(1)} \nabla_y \ell(y^{(1)}, t^{(1)}) \\ \vdots \\ b_k^{(N)} \nabla_y \ell(y^{(N)}, t^{(N)}) \end{bmatrix}, \quad b_k^{(j)} \overset{\text{i.i.d.}}{\sim} \text{Ber}(\frac{\mathcal{B}}{N}). \tag{12}$$

It is straightforward to show that eq. (12) is an unbiased estimator of the output-space gradient, i.e. $\mathbb{E}_{b_k^{(1)},\ldots,b_k^{(N)}}[\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})] = \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. To map eq. (12) to the parameter space, we can simply multiply it with the Jacobian transpose $\tilde{\nabla}^{\mathcal{B}}_{\theta} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{t}; \theta_k) = J^\top \tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. Ziyin et al. (2022) have shown that the covariance of $\tilde{\nabla}^{\mathcal{B}}_{\theta} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{t}; \theta_k)$ contains a term that is proportional to the loss, which characterizes the key property of minibatch noise. One of our main contributions is to tractably incorporate the minibatch noise model in eq. (12) into our model in eq. (11).

For convenience, we rewrite eq. (12) using indicator matrices

$$\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) = \frac{N}{\mathcal{B}} (\sum_{j=1}^{N} b^{(j)} \mathbb{1}_{jj}) \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}), \tag{13}$$

where $\mathbb{1}_{jj} \in \mathbb{R}^{N \times N}$ is a matrix with all elements equal to 0, except that the (j, j)-th element is 1. In eq. (11a), replacing the full-batch gradient signal $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$ with the minibatched version $\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$, we have

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}J^\top \underbrace{\left(\frac{N}{\mathcal{B}} (\sum_{j=1}^{N} b^{(j)} \mathbb{1}_{jj}) \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) \right)}_{\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})} + \bar{B}J^\top \epsilon_k \tag{14a}$$

$$\boldsymbol{y}_k = J\bar{C}\xi_k. \tag{14b}$$

Equation (14a) captures the idea of our minibatch gradient model. However, as we will discuss in Section 3.4 and Section 3.5, our framework requires construct and numerically solve SDP problems similar to Theorem 1. It is not computationally tractable to directly do so using eq. (14a), as the dimensions of quantities are too large. In the following section, we develop dimension reduction techniques to render eq. (14) in a computationally tractable form.

3.2 Dimensionality reduction 1: weight-space projection

The optimizer-related matrices $\bar{A}, \bar{B}, \bar{C}$ are in the weight space, and their dimensions are multiples of d, where d is the number of model parameters (e.g. for gradient descent, $\bar{A}, \bar{B}, \bar{C} \in \mathbb{R}^{d\times d}$, and for heavy-ball and NAG, $\bar{A} \in \mathbb{R}^{2d\times 2d}$, $\bar{B} \in \mathbb{R}^{2d\times d}$, $\bar{C} \in \mathbb{R}^{d\times 2d}$). The Jacobian J has dimension $N \times d$, where N is the number of data points optimized on.

For a machine learning problem, both N and d are typically large. If we naively use eq. (14) to construct our convex program for certifying convergence, the computational cost would be prohibitive. The number of variables in the SDP problem would be quadratic in d, and the matrices with dimensions $N \times d$ or $d \times d$ could be too large to even store in memory. To address this problem, we propose two dimensionality reduction techniques: weight-space projection and Jacobian binning.

Weight-space projection Ideally, we do not want the size of our convex program to grow with the size of training data. Conveniently, we note that the data dimension is always associated with a summation, due to the definition of the loss function (eq. (1)). This indicates that we can remove the data dimension N from our state-space model eq. (14) by projecting the associated quantities onto the weight space. In fact, this projection naturally exists in eq. (14): any quantity with dimension N is always multiplied with $J^\top$. Assuming $N \geq d$, we perform singular value decomposition (SVD) on J

$$J = USV^\top, \quad U \in \mathbb{R}^{N\times d},\ S, V \in \mathbb{R}^{d\times d}. \tag{15}$$

We will project all quantities of dimension N using the matrix U, and define the projected quantities as

$$\mathtt{y} := U^\top \boldsymbol{y} \in \mathbb{R}^d, \quad \varepsilon := U^\top \epsilon \in \mathbb{R}^d, \quad \mathtt{J} := U^\top J \in \mathbb{R}^{d\times d}, \quad \nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) := U^\top \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) \in \mathbb{R}^d.$$

To rewrite eq. (14) using the projected quantities, we still need to apply the projection $U^\top$ to the minibatch gradient $\tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. We have

$$\tilde{\nabla}^{\mathcal{B}}_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) := U^\top \tilde{\nabla}^{\mathcal{B}}_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) = \frac{N}{\mathcal{B}} \sum_{j=1}^{N} b^{(j)} U^\top \mathbb{1}_{jj} \nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}). \tag{16}$$

Equation (16) still contains the term $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$, which has dimension N. Instead, we would like to express eq. (16) in terms of the projected quantity $\nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. To do this, we need to make the assumption that the output-space gradient lies in the column space of U.

Assumption 1. $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) \in col(U)$.

This is a reasonable assumption, as we are only interested in the component of $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$ that has an effect on updating θ. A detailed justification is provided in appendix C.

With this assumption, we proceed to express $\nabla_{\boldsymbol{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$ in terms of $\nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t})$. With the derivation detailed in appendix C, we have

$$\tilde{\nabla}^{\mathcal{B}}_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) = \nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) - \frac{N}{\mathcal{B}} \sum_{j=1}^{N} r^{(j)} u_j u_j^\top \nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}), \tag{17}$$

where u_j is the j-th column of U, and $r^{(j)} = \frac{\mathcal{B}}{N} - b^{(j)}$ is the Bernoulli random variable with zero mean

$$r^{(j)} = \begin{cases} \frac{\mathcal{B}}{N} - 1, & \text{w. prob } \frac{\mathcal{B}}{N} \\ \frac{\mathcal{B}}{N}, & \text{w. prob } 1 - \frac{\mathcal{B}}{N} \end{cases}.$$

Equation (14a) can now be rewritten with the quantities in eq. (15) and eq. (17)

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}\mathtt{J}^\top \nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) - \sum_{j=1}^{N} \big(r^{(j)} \frac{N}{\mathcal{B}} \bar{B}\mathtt{J}^\top u_j u_j^\top\big) \nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) + \bar{B}\mathtt{J}^\top \varepsilon_k \tag{18a}$$

$$\mathtt{y}_k = \mathtt{J}\bar{C}\xi_k. \tag{18b}$$

For NQM, we have $\nabla_{\mathtt{y}} L(\boldsymbol{y}_k, \boldsymbol{t}) = \mathtt{y}_k$. Substituting eq. (18b) into eq. (18a), we have

$$\xi_{k+1} = (\underbrace{\bar{A} + \bar{B}\mathtt{J}^\top \mathtt{J}\bar{C}}_{\mathtt{A}})\xi_k + \sum_{j=1}^{N} \Big(\underbrace{-\frac{N}{\mathcal{B}} \bar{B}\mathtt{J}^\top u_j u_j^\top \mathtt{J}\bar{C}}_{\mathtt{A}_j} \Big) \xi_k \cdot r^{(j)} + \underbrace{\bar{B}\mathtt{J}^\top}_{\mathtt{B}_\omega} \varepsilon_k \tag{19a}$$

$$\mathtt{y}_k = \mathtt{J}\bar{C}\xi_k. \tag{19b}$$

3.3 Dimensionality reduction 2: Jacobian binning

In Section 3.2, we have reduced the output dimension N to the weight space dimension d. However, in practice, d can still be too large for the convex program that we will construct. In this section, we introduce an additional dimensionality reduction technique — Jacobian binning.

First, we will find it convenient to reduce the projected Jacobian $\mathtt{J}$ into a diagonal matrix. We make the following assumption that the matrix V of the Jacobian SVD is identity.

Assumption 2. $V = I$.

Assumption 2 is without loss of generality for the optimizers we consider, which are invariant to the choice of basis of the network parameters. For example, consider the heavy-ball and NAG optimizers. If we reparameterize the model by projecting θ_k to a different basis with some orthogonal projection, apply the optimizer updates from the new basis, and project the result back to the original basis, the training dynamics would be identical to applying the optimizer updates from the original basis. Assumption 2 holds as long as $\bar{A}, \bar{B}, \bar{C}$ are obtained from Kronecker products with the $d \times d$ identity matrix, which is the case for all first-order optimizers that are uniformly applied to all weight parameters.

With this assumption, we have $\mathtt{J} = S$. Note that it doesn't mean that neural network training is fully decoupled in the weight space, because minibatching introduces coupling between the dynamics of different weight parameters (the $u_j u_j^\top$ matrices in eq. (19a) are not diagonal). This is a key difference from NQM (Zhang et al., 2019), where each weight parameter is assumed to evolve independently.

To further reduce the size of our convex program, we group the Jacobian singular values into bins. Let the singular values of $\mathtt{J}$ be $s_1, \ldots, s_d$. It is observed that the Hessian spectrum of neural networks often follows the power law (Zhang et al., 2019; Martin & Mahoney, 2021). Therefore, we make the following approximation to reduce the dimensionality of the Jacobian.

Approximation 3 (Jacobian binning)**.** *We partition the singular values into m bins ($m \ll d$), with values $\bar{s}_1, \ldots, \bar{s}_m$ arranged in logspace. We group s_i into the j-th bin, if $\bar{s}_j \le s_i < \bar{s}_{j-1}$ ($\bar{s}_j \le s_i$ if $j = 1$), and let the size of the i-th bin be n_i. Since we have assumed without loss of generality that $\mathtt{J}$ is diagonal, we denote the reduced Jacobian as $\mathtt{J}_{bin} = diag(\bar{s}_1, \ldots, \bar{s}_m)$. In simulations, we replace the Jacobian $\mathtt{J}$ with $\mathtt{J}_{bin}$. Related matrices containing dimension d are scaled down accordingly.*[2]

With Approximation 3, we lose the fine-grained information, but capture the overall distribution of the singular values. This is a lossy but reasonable approximation to the Jacobian, and is also adopted in Zhang et al. (2019). The experiments in Zhang et al. (2019) suggest that NQM with Jacobian binning produces predictions that match real neural network training dynamics.

3.4 Certifying convergence of NQM with minibatch noise

With the dimensionality reduction techniques in Sections 3.2 and 3.3, we can now establish the conditions that allow us to certify convergence of NQM under the minibatch model in Section 3.1. We first restate the state-space representation in eq. (19):

[2]This contains $\bar{A}, \bar{B}, \bar{C}$ and matrices that are products and / or projections of those. To scale down, simply reduce the identity and zero matrices to $m \times m$ in their definitions, which are similar to eq. (6).

State-space representation for an NQM with minibatch noise

$$\xi_{k+1} = \mathtt{A}\xi_k + \sum_{j=1}^{N} \mathtt{A}_j \xi_k \cdot r^{(j)} + \mathtt{B}_\omega \varepsilon_k \tag{20a}$$

$$\mathtt{y}_k = \mathtt{J}\bar{C}\xi_k \tag{20b}$$

where $r_k^{(j)}$ are i.i.d. zero-centered Bernoulli random variables, ε_k is the Gaussian noise with covariance $\frac{1}{\mathcal{B}}I$, i.i.d. sampled at each step and projected to the weight space (Section 3.2), and

$$\mathtt{A} = \bar{A} + \bar{B}\mathtt{J}^\top \mathtt{J}\bar{C}, \tag{20c}$$

$$\mathtt{A}_j = -\frac{N}{\mathcal{B}}\bar{B}\mathtt{J}^\top u_j u_j^\top \mathtt{J}\bar{C}, \text{ for } j = 1, \ldots, N \tag{20d}$$

$$\mathtt{B}_\omega = \bar{B}\mathtt{J}^\top \tag{20e}$$

Theorem 2 states the LMI conditions for certifying the asymptotic convergence rate of an NQM with minibatch noise.

Theorem 2 (Asymptotic convergence rate of NQM with minibatch noise)**.** *Consider the system in Equation* (20)*, where the label noise ε_k and the minibatch noise $r_k^{(j)}$ satisfy:*

$$\mathbb{E}[\varepsilon_k] = 0, \qquad \mathbb{E}[r_k^{(j)}] = 0,$$
$$\mathbb{E}[\varepsilon_k \varepsilon_\tau^\top] = 0 \ \forall k \neq \tau, \qquad r_k^{(j)} \text{ are i.i.d.},$$
$$\mathbb{E}[\varepsilon_k \varepsilon_k^\top] = \frac{1}{\mathcal{B}}I, \qquad \mathbb{E}[r_k^{(j)} r_k^{(j)}] = \sigma_j^2.$$

If there exists $P \succ 0$ and $0 < \rho < 1$ such that:

$$\mathtt{A}^\top P \mathtt{A} - \rho^2 P + \sum_{j=1}^{N} \sigma_j^2 \mathtt{A}_j^\top P \mathtt{A}_j \preceq 0 \tag{21}$$

Then the expected loss $\frac{1}{2}\mathbb{E}[\mathtt{y}_k^\top \mathtt{y}_k]$ asymptotically converges to a steady-state risk with rate ρ^2.

The proof for Theorem 2 can be found in Appendix D.

3.5 Certifying convergence under network nonlinearity

We have so far dealt with NQM, which corresponds to a linear model with quadratic loss. For an NQM, the gradient g is a linear function of parameters θ plus noise (eq. (3)). However, this no longer holds for neural networks training. In order to model more realistic training dynamics, we need to model the nonlinearity relationship between the gradient g and parameters θ.

Since we use the squared-error loss, the output-space gradient is always linear with respect to the output: $\nabla_y L(\boldsymbol{y}_k, \boldsymbol{t}) = y_k$. The source of the nonlinearity in g_k comes from the fact that the Jacobian $\mathtt{J}_k$ at step k is dependent on θ_k. This is in contrast to the NQM, where the Jacobian is constant.

However, directly modeling the Jacobian $\mathtt{J}_k$ as a function of θ_k would be too complex, and would not lead to any computationally tractable analysis. Instead, we adopt an alternative paradigm inspired by robust control theory. In this paradigm, we capture the nonlinear training dynamics by allowing the Jacobian $\mathtt{J}_k$ to vary within an uncertainty set, which can be described by quadratic constraints and formulated as part of an SDP problem. Specifically, we model the uncertainty set for $\mathtt{J}_k$ as a set of matrices with multiplicative perturbations on the constant matrix $\mathtt{J}$:

$$\mathtt{J}_k \in \mathcal{J}, \quad \text{where } \mathcal{J} = \{\mathtt{J}' : \mathtt{J}' = (I + M_k)\mathtt{J}(I + N_k)^\top, \|M_k\| \leq \delta_M \ll 1, \|N_k\| \leq \delta_N \ll 1\}. \tag{22}$$

$M_k, N_k \in \mathbb{R}^{d\times d}$ represent "small" multiplicative perturbations. We assume that with properly chosen bounds for the perturbation size δ_M and δ_N, the true Jacobian of the nonlinear model lies within the uncertainty set $\mathcal{J}$ at all times. The worst-case convergence analysis over the family of models described by eq. (22) is tractable, and can be formulated as an SDP problem.

We consider multiplicative perturbations, because the singular values of the Jacobian vary greatly[3], and we would like the relative size of the perturbation in different singular directions to be balanced. A purely additive perturbation (e.g. $\mathtt{J}_k = \mathtt{J} + \Delta\mathtt{J}_k$), in contrast, would have little relative effect on the directions with large singular values, but much larger relative effect on the directions with small singular values. Therefore, we favor multiplicative to additive perturbations, though if needed, our framework is easily extendable to accommodate an additional additive perturbation.

To simplify notations, we express eq. (22) simply as $\mathtt{J}_k = (I + M_k)\mathtt{J}(I + N_k)^\top$. Expanding the product, notice that we can drop the second-order terms, when δ_M and δ_N are assumed to be small:

$$\mathtt{J}_k = \mathtt{J} + M_k\mathtt{J} + \mathtt{J}N_k^\top + \mathcal{O}(\delta_M\delta_N). \tag{23}$$

In our model for nonlinearity, we only apply the Jacobian perturbation (eq. (23)) to the matrix $\mathtt{A}$ in eq. (20). For simplicity, we do not apply the perturbation to the terms associated with minibatch noise and the label noise (i.e. $\mathtt{A}_j$, $j = 1, \ldots, N$ and $\mathtt{B}_\omega$). Future work could investigate how such perturbations can be applied to the noise terms while keeping the SDP formulation tractable.

Denote $\mathtt{A}_k = \bar{A} + \bar{B}\mathtt{J}_k^\top \mathtt{J}_k\bar{C}$ as the matrix $\mathtt{A}$ with the Jacobian perturbation at step k. Substituting in eq. (23), we have

$$\begin{aligned}
\mathtt{A}_k &= \bar{A} + \bar{B}(\mathtt{J} + M_k\mathtt{J} + \mathtt{J}N_k^\top)^\top(\mathtt{J} + M_k\mathtt{J} + \mathtt{J}N_k^\top)\bar{C} + \mathcal{O}(\delta_M\delta_N) && \text{(24a)}\\
&= \mathtt{A} + \bar{B}\mathtt{J}^\top(M_k + M_k^\top)\mathtt{J}\bar{C} + \bar{B}N_k\mathtt{J}^\top\mathtt{J}\bar{C} + \bar{B}\mathtt{J}^\top\mathtt{J}N_k^\top\bar{C} + \mathcal{O}(\delta_M\delta_N + \delta_M^2 + \delta_N^2), && \text{(24b)}\\
&\approx \mathtt{A} + \bar{B}\mathtt{J}^\top(M_k + M_k^\top)\mathtt{J}\bar{C} + \bar{B}N_k\mathtt{J}^\top\mathtt{J}\bar{C} + \bar{B}\mathtt{J}^\top\mathtt{J}N_k^\top\bar{C}, && \text{(24c)}
\end{aligned}$$

where in eq. (24c), we used the fact that δ_M and δ_N are small, and dropped second or higher-order terms.

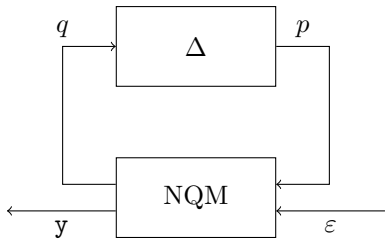


Figure 1: Diagram for the interleaved system with the NQM base model (with minibatch noise) and the Jacobian perturbation (the Δ block). With Jacobian perturbation, we can model the nonlinearity in the training dynamics.

Eventually, we would like to certify the stability of the whole system. In order to do that, we first separate eq. (24c) into two interleaved blocks, as illustrated in Figure 1. The Jacobian perturbation is represented by the Δ block.[4] The Δ block takes in signals q, applies perturbation, and outputs p. The perturbed signals p are then supplied to the NQM base block as input. This is clearly shown in the state-space representation below.

[3]In neural networks, eigenvalues of the Hessian spectrum roughly follows the power law (proportional to $\frac{1}{i}$, where i is the index of the eigenvalue) (Zhang et al., 2019). With the Gauss-Newton Hessian approximation, the singular values of the Jacobian are roughly proportional to $\frac{1}{\sqrt{i}}$.

[4]In fact, Δ contains three sub-blocks, one for each of the three terms in eq. (24c) with Jacobian perturbation.

Combined System

The combined system with Jacobian uncertainty and minibatch noise is

$$\xi_{k+1} = \mathtt{A}\xi_k + \sum_{j=1}^{N} \mathtt{A}_j \xi_k \cdot r_k^{(j)}$$

$$\bar{B}\mathtt{J}^\top \underbrace{(M_k + M_k^\top)\overbrace{\mathtt{J}\bar{C}\xi_k}^{q_{1,k}}}_{p_{1,k}} + \bar{B}\underbrace{N_k \overbrace{\mathtt{J}^\top \mathtt{J}\bar{C}\xi_k}^{q_{2,k}}}_{p_{2,k}} + \bar{B}\mathtt{J}^\top \mathtt{J}\underbrace{N_k^\top \overbrace{\bar{C}\xi_k}^{q_{3,k}}}_{p_{3,k}} + \mathtt{B}_\omega \varepsilon_k \tag{25a}$$

$$\mathtt{y}_k = \mathtt{J}\bar{C}\xi_k, \tag{25b}$$

where $\mathtt{A}$, $\mathtt{A}_j$, $\mathtt{B}_\omega$ are defined in Equation (20). The input and output quantities are

$$q_{1,k} = \mathtt{J}\bar{C}\xi_k \qquad p_{1,k} = \Delta_1 q_{1,k} \quad (\Delta_1 = M_k + M_k^\top) \tag{25c}$$

$$q_{2,k} = \mathtt{J}^\top \mathtt{J}\bar{C}\xi_k \qquad p_{2,k} = \Delta_2 q_{2,k} \quad (\Delta_2 = N_k) \tag{25d}$$

$$q_{3,k} = \bar{C}\xi_k \qquad p_{3,k} = \Delta_3 q_{3,k} \quad (\Delta_3 = N_k^\top). \tag{25e}$$

We can rewrite eq. (25) in matrix form,

$$\left[\begin{array}{c} \xi_{k+1} \\ \hline q_{1,k} \\ q_{2,k} \\ q_{3,k} \\ \hline \mathtt{y}_k \end{array}\right] = \left[\begin{array}{c|ccc|ccc|c} \mathtt{A} & \mathtt{A}_1 & \cdots & \mathtt{A}_N & \bar{B}\mathtt{J}^\top & \bar{B} & \bar{B}\mathtt{J}^\top\mathtt{J} & \mathtt{B}_w \\ \hline \mathtt{J}\bar{C} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \mathtt{J}^\top\mathtt{J}\bar{C} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \bar{C} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \hline \mathtt{J}\bar{C} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} \xi_k \\ \hline r_k^{(1)} \\ \vdots \\ r_k^{(N)} \\ \hline p_{1,k} \\ p_{2,k} \\ p_{3,k} \\ \hline \varepsilon_k \end{array}\right]. \tag{26}$$

To simplify notations, we group certain quantities in eq. (26) together and define

$$\hat{A} = \mathtt{A}, \quad \hat{B} = \begin{bmatrix} \bar{B}\mathtt{J}^\top & \bar{B} & \bar{B}\mathtt{J}^\top\mathtt{J} \end{bmatrix}, \tag{27a}$$

$$\hat{C}_1 = \mathtt{J}\bar{C}, \quad \hat{C}_2 = \mathtt{J}^\top\mathtt{J}\bar{C}, \quad \hat{C}_3 = \bar{C}, \quad \hat{C}_\mathtt{y} = \mathtt{J}\bar{C}, \tag{27b}$$

$$\hat{D}_1 = \hat{D}_2 = \hat{D}_3 = 0 \in \mathbb{R}^{d\times d}. \tag{27c}$$

Then we rewrite eq. (26) as

$$\left[\begin{array}{c} \xi_{k+1} \\ \hline q_{1,k} \\ q_{2,k} \\ q_{3,k} \\ \hline \mathtt{y}_k \end{array}\right] = \left[\begin{array}{c|ccc|ccc|c} \hat{A} & \mathtt{A}_1 & \cdots & \mathtt{A}_N & & \hat{B} & & \mathtt{B}_w \\ \hline \hat{C}_1 & 0 & \cdots & 0 & \hat{D}_1 & 0 & 0 & 0 \\ \hat{C}_2 & 0 & \cdots & 0 & 0 & \hat{D}_2 & 0 & 0 \\ \hat{C}_3 & 0 & \cdots & 0 & 0 & 0 & \hat{D}_3 & 0 \\ \hline \hat{C}_\mathtt{y} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} \xi_k \\ \hline r_k^{(1)} \\ \vdots \\ r_k^{(N)} \\ \hline p_{1,k} \\ p_{2,k} \\ p_{3,k} \\ \hline \varepsilon_k \end{array}\right]. \tag{28}$$

To certify the stability of the combined system (eq. (25) and fig. 1), we need to constrain the uncertainty in the Δ block. Intuitively, the quantities $\Delta_1, \Delta_2, \Delta_3$ represent the perturbation of the Jacobian at each step, and that they should be "small" in some sense, in order for the whole system to remain stable. In this paper, we apply ℓ_2 constraints on the $\Delta_1, \Delta_2, \Delta_3$ blocks

$$\|\Delta_1\|_2 \le \delta_1, \quad \|\Delta_2\|_2 \le \delta_2, \quad \|\Delta_3\|_2 \le \delta_3, \quad \delta_1, \delta_2 \ge 0, \quad \delta_2 = \delta_3. \tag{29}$$

Note that since $\Delta_2 = N_k$, $\Delta_3 = N_k^\top$, we set $\delta_2 = \delta_3$ without loss of generality.

To make the constraints in eq. (29) compatible with the semidefinite programming (SDP) formulation, we express eq. (29) as quadratic constraints on the input and output quantities p, q. Equation (29) can be equivalently written as

$$\begin{bmatrix} q_{1,k} \\ p_{1,k} \end{bmatrix}^\top \begin{bmatrix} \delta_1^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} q_{1,k} \\ p_{1,k} \end{bmatrix} \geq 0, \quad \begin{bmatrix} q_{2,k} \\ p_{2,k} \end{bmatrix}^\top \begin{bmatrix} \delta_2^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} q_{2,k} \\ p_{2,k} \end{bmatrix} \geq 0, \quad \begin{bmatrix} q_{3,k} \\ p_{3,k} \end{bmatrix}^\top \begin{bmatrix} \delta_2^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} q_{3,k} \\ p_{3,k} \end{bmatrix} \geq 0. \tag{30}$$

Now, we are ready to state the conditions for certifying the convergence rate of the combined system, with minibatch noise and Jacobian perturbation.

Theorem 3 (Certifying convergence rate for the combined model)**.** *Consider the system in Equation* (28) *where the noise signals ε_k and $r_k^{(j)}$ satisfy:*

$$\begin{aligned} &\mathbb{E}[\varepsilon_k] = 0, && \mathbb{E}[r_k^{(j)}] = 0, \\ &\mathbb{E}[\varepsilon_k \varepsilon_\tau^\top] = 0 \ \forall k \neq \tau, && r_k^{(j)} \text{ are i.i.d.}, \\ &\mathbb{E}[\varepsilon_k \varepsilon_k^\top] = I, && \mathbb{E}[r_k^{(j)} r_k^{(j)}] = \sigma_j^2. \end{aligned}$$

Given $\delta_1, \delta_2 \geq 0$ and $\delta_2 = \delta_3$, assume that operators Δ_l $(l = 1, 2, 3)$ (defined in eq. (25)*) satisfy*

$$\begin{bmatrix} q_{l,k} \\ \Delta_l q_{l,k} \end{bmatrix}^\top \begin{bmatrix} \delta_l^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} q_{l,k} \\ \Delta_l q_{l,k} \end{bmatrix} \geq 0, \ \forall q_{l,k} \in \mathbb{R}^d, \tag{31}$$

if there exists a positive-definite matrix P, $0 < \rho < 1$, and $\lambda_1, \lambda_2, \lambda_3 \geq 0$ such that:

$$\begin{aligned} &\begin{bmatrix} I & 0 \\ \hat{A} & \hat{B} \end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & 0 \\ 0 & P \end{bmatrix} \begin{bmatrix} I & 0 \\ \hat{A} & \hat{B} \end{bmatrix} + \sum_{j=1}^{N} \sigma_j^2 \begin{bmatrix} \mathtt{A}_j & 0 \end{bmatrix}^\top P \begin{bmatrix} \mathtt{A}_j & 0 \end{bmatrix} \\ &\quad + \lambda_1 \begin{bmatrix} \hat{C}_1 & \hat{D}_1 & 0 & 0 \\ 0 & I & 0 & 0 \end{bmatrix}^\top \begin{bmatrix} \delta_1^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} \hat{C}_1 & \hat{D}_1 & 0 & 0 \\ 0 & I & 0 & 0 \end{bmatrix} \\ &\quad + \lambda_2 \begin{bmatrix} \hat{C}_2 & 0 & \hat{D}_2 & 0 \\ 0 & 0 & I & 0 \end{bmatrix}^\top \begin{bmatrix} \delta_2^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} \hat{C}_2 & 0 & \hat{D}_2 & 0 \\ 0 & 0 & I & 0 \end{bmatrix} \\ &\quad + \lambda_3 \begin{bmatrix} \hat{C}_3 & 0 & 0 & \hat{D}_3 \\ 0 & 0 & 0 & I \end{bmatrix}^\top \begin{bmatrix} \delta_3^2 I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} \hat{C}_3 & 0 & 0 & \hat{D}_3 \\ 0 & 0 & 0 & I \end{bmatrix} \preceq 0, \end{aligned} \tag{32}$$

then the expected loss $\frac{1}{2}\mathbb{E}[\mathtt{y}_k^\top \mathtt{y}_k]$ asymptotically converges to a steady-state risk with rate ρ^2.

The proof of Theorem 3 can be found in Appendix D.

4 Simulations

We use the MOSEK solver (ApS, 2025) via the CVXPY interface (Diamond & Boyd, 2016) to solve the semidefinite program (SDP) defined in theorem 3. We set the size of the training set to $N = 2000$, and the parameter space dimension to $d = 1000$. The Jacobian singular values are set to $\{\frac{1}{\sqrt{i}}\}_{i=1}^d$, discretized into 10 bins logarithmically. The Jacobian singular values are set such that the eigenvalues of the Hessian roughly follows a power law $\{\frac{1}{i}\}_{i=1}^d$, which is observed empirically in neural networks (Zhang et al., 2019). Additional simulation details are included in appendix E. We use bisection search to find the smallest ρ (up to an error of 10^{-12}) such that the SDP is feasible.

4.1 Quadratic model with minibatch noise

First, we consider the quadratic model with minibatch noise. In this case, there is no uncertainty in the model Jacobian ($\delta_1 = \delta_2 = \delta_3 = 0$). This scenario is similar to the noisy quadratic model in Zhang et al.

(2019), but with important differences: 1) we model minibatch training as multiplicative noise (Section 3.1), as opposed to an additive gradient noise in Zhang et al. (2019); 2) in addition to the classic (heavy-ball) momentum, we also analyze the Nesterov accelerated gradient (NAG), which is not considered in Zhang et al. (2019) due to its lack of closed-form solution for the training dynamics. 3) we focus on the asymptotic performance, as opposed to finite-step performance in Zhang et al. (2019). We discuss several insights from the results.

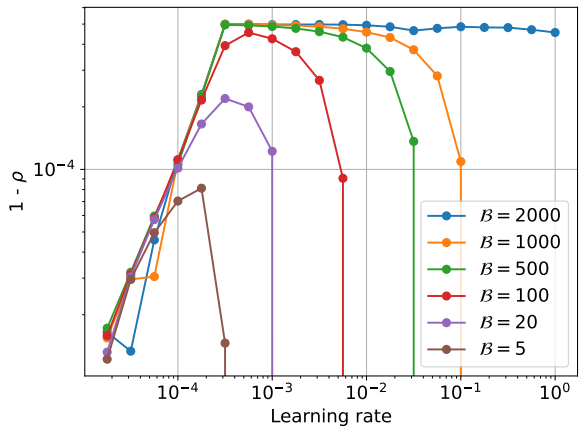
Insight 1: heavy-ball momentum is fragile and inefficient at small batch sizes. With a small batch size, heavy-ball is forced to use a small learning rate, resulting in worse convergence rate. This effect worsens with larger momentum values.

The NQM (Zhang et al., 2019) predicts that, as batch size increases, the optimal learning rate first proportionally increases with respect to batch size, resulting in perfect scaling of convergence rate and batch size (increasing the batch size by a factor of k leads to a k-fold decrease in the steps taken to reach a loss threshold). However, this perfect scaling regime ends at a certain batch size $\mathcal{B}_{\text{critical}}$. At batch size $\mathcal{B}_{\text{critical}}$, the optimal learning rate is at the critical damping threshold. Further increasing the learning rate would lead to no improvement in convergence rate, resulting in diminishing returns when batch size is scaled up beyond $\mathcal{B}_{\text{critical}}$. This is due to the fact that the optimization enters the curvature-dominated regime, where large learning rates lead to oscillations, characteristic of an underdamped system. According to the NQM, when increasing the learning rate in the curvature-dominated regime, the convergence rate holds constant. In full-batch training ($\mathcal{B} = N$), this matches the prediction of our framework (the "$\mathcal{B} = 2000$" curve in Figure 2a). However, we find that in minibatch training, there is an optimal learning rate for heavy-ball momentum, beyond which the convergence rate quickly degrades, and instability may eventually occur. The performance degradation at large learning rates worsens with smaller batch sizes. This effect is more pronounced at large momentum values (Figures 2a, 11a and 12a show $\beta = 0.999, 0.99, 0.9$ respectively).

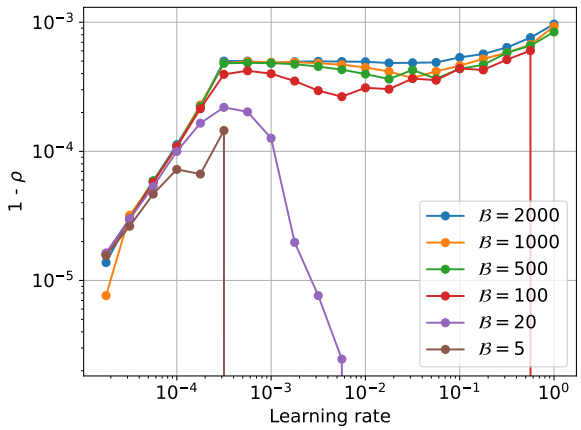
The NQM fails to predict heavy-ball's fragility and inefficiency to smaller batch sizes due to its inaccurate minibatch noise modeling. Our insight agrees with the analysis in Lee et al. (2022), who studies the different convergence regimes of heavy-ball momentum for a Gaussian random least squares problem. It also agrees with the analysis in Bollapragada et al. (2025) suggesting that heavy-ball momentum only achieves acceleration with a sufficiently large batch size. Unlike the NQM, our LMI framework is able to predict this effect due to proper modeling of the minibatch noise.

Insight 2: in minibatch training, NAG is more robust than heavy-ball momentum at large learning rates.

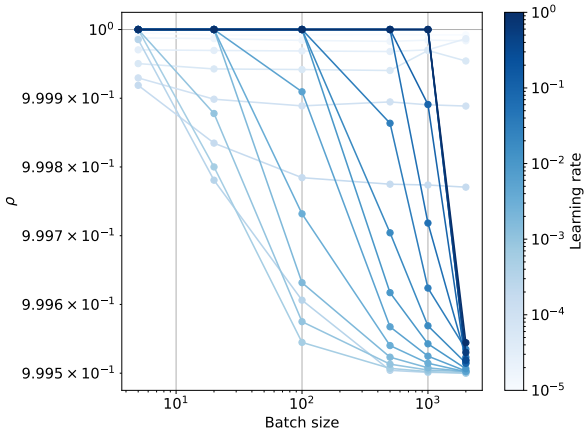
In contrast to heavy-ball momentum, our LMI framework predicts that NAG is much more robust at large learning rates. Figure 2b shows that, even with a very large momentum value ($\beta = 0.999$), for a reasonably large batch size (such as 100), increasing the learning rate past the critical damping threshold does not hurt the convergence rate. Figure 3 plots the learning rate that achieves the best convergence rate. We see that compared to HB, NAG can benefit from a larger learning rate with large momentum values, whereas HB is forced to use a smaller learning rate to avoid instability. This prediction agrees with practical observations and analysis in other works. Sutskever et al. (2013) noted that the difference of HB and NAG is only distinct when the learning rate is reasonably large, and that NAG behaves more stably than HB in many situations. Sutskever et al. (2013) argues that the NAG's comparative stability is due to the fact that it changes the momentum in a quicker and more responsive way. Rather than relying on heuristics, our framework provides a principled and quantitative explanation. We also empirically confirm the comparative robustness of NAG through a sweep of batch sizes and learning rates for the MNIST task. Figure 5 shows the training loss of a linear model on MNIST dataset with mean squared error loss and $\beta = 0.999$. Compared to HB, NAG remains stable at larger learning rates and batch sizes.



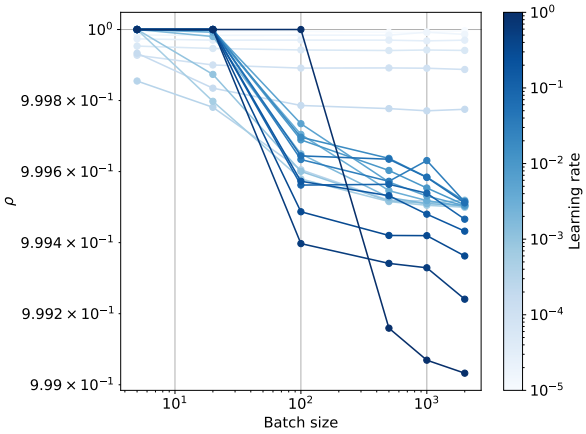
(a) Convergence rate vs. learning rate (HB, $\beta = 0.999$)



(b) Convergence rate vs. learning rate (NAG, $\beta = 0.999$)



(c) Convergence rate vs. batch size (HB, $\beta = 0.999$)



(d) Convergence rate vs. batch size (NAG, $\beta = 0.999$)

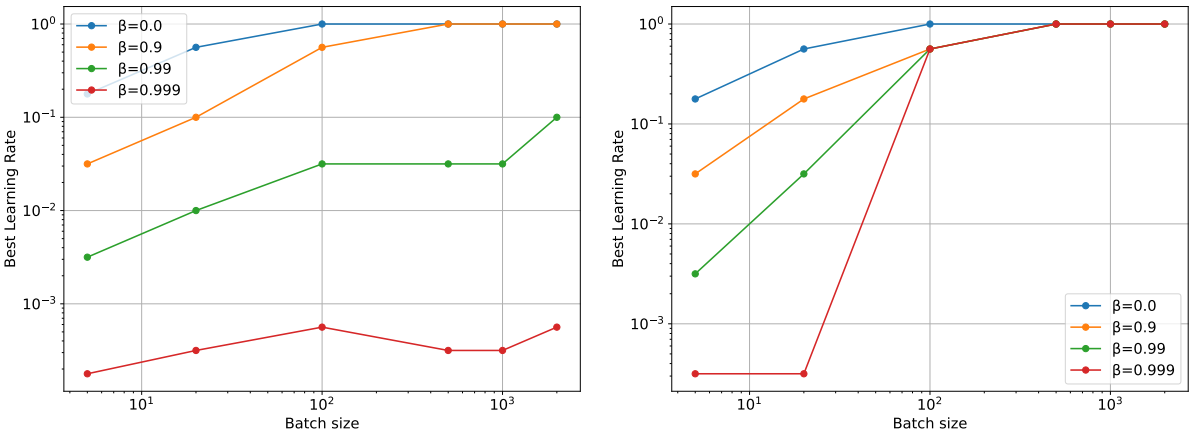
Figure 2: **With large momentum, NAG is more robust than HB at larger learning rates at small batch sizes.** With large momentum and learning rates, the system becomes overdamped.

4.2 Nonlinear training dynamics

> Insight 3: NAG is more robust to the amount of nonlinearity in training dynamics than heavy-ball momentum.

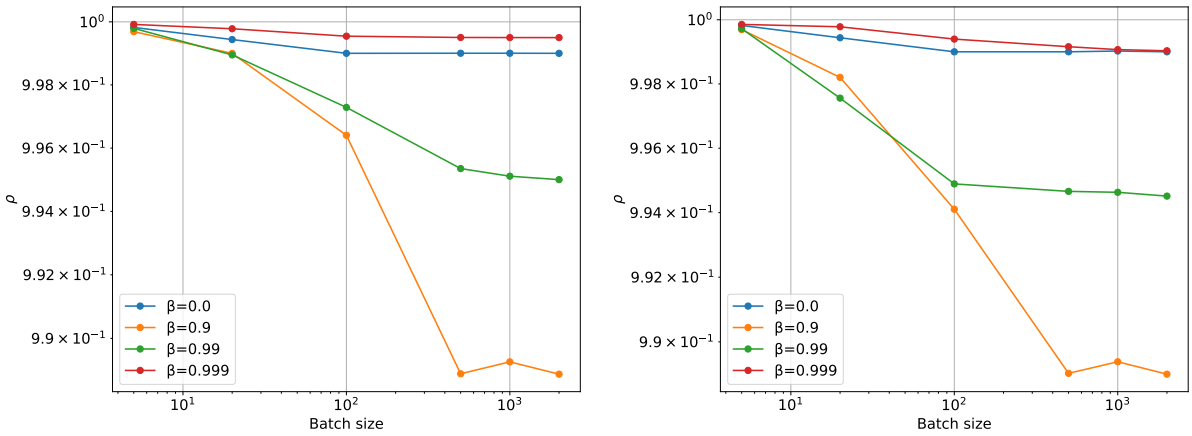
We quantify HB and NAG's robustness to nonlinear training dynamics using our Jacobian uncertainty modeling in Section 3.5. In Figure 6, we show the convergence rate as a function of Jacobian uncertainty. Compared to heavy-ball momentum, we can certify a better convergence rate for NAG under model uncertainty, both for the full-batch (Figure 6) and minibatch (Figure 13) cases. This insight adds to the previous arguments for robustness of NAG.

We empirically confirm this insight through neural network training on the MNIST dataset. Figure 7 shows heatmaps of the training loss plotted against learning rates and batch sizes. We train a multi-layer perceptron (MLP) model on MNIST dataset with mean squared error loss and $\beta = 0.999$. The MLP model has 2 hidden layers with 1000 hidden units each and ReLU activation (Nair & Hinton, 2010). In order to demonstrate a varying amount of nonlinearity in training dynamics, we present the results with different parameterizations. Figures 7a and 7b show the heatmaps for the standard parameterization (SP) and the Maximal Update Parameterization (μP), respectively (Yang et al., 2022). SP and μP for an MLP differ in



(a) Heavy Ball (b) NAG

Figure 3: **NAG allows larger learning rates than heavy-ball momentum.** Optimal learning rate vs. batch size in NQM with minibatch noise, for heavy-ball momentum and NAG, for different momentum values. With large momentum values, the optimal learning rate for NAG still scales well with batch size, whereas heavy-ball momentum suffers from instability at large learning rates.



(a) Heavy Ball (b) NAG

Figure 4: Convergence rate of heavy-ball and NAG with different momentum values.

the per-layer learning rates and initialization scales, and lead to different training dynamics. For wide neural networks, SP training dynamics is relatively linear, as it falls into the kernel regime in the infinite-width limit (Lee et al., 2019). In contrast, μP leads to maximal feature learning in the infinite-width limit, with more nonlinear training dynamics (Yang & Hu, 2021). The results show that compared to HB, NAG remains stable at larger learning rates and batch sizes when training nonlinearity is present. This observation is more pronounced for μP than SP, suggesting that NAG is more robust to training nonlinearity.

We note that it is difficult to directly verify the sensitivity to Jacobian uncertainty (as in Figure 6) with experiments, because 1) the Jacobian uncertainty is difficult to quantify empirically, as our uncertainty model (eq. (23)) is underspecified; and 2) it is difficult to precisely control the amount of Jacobian uncertainty during training. Our results in Figure 7 offer two discrete points on heatmaps such as Figure 6, and serve as supporting evidence for insight 3.

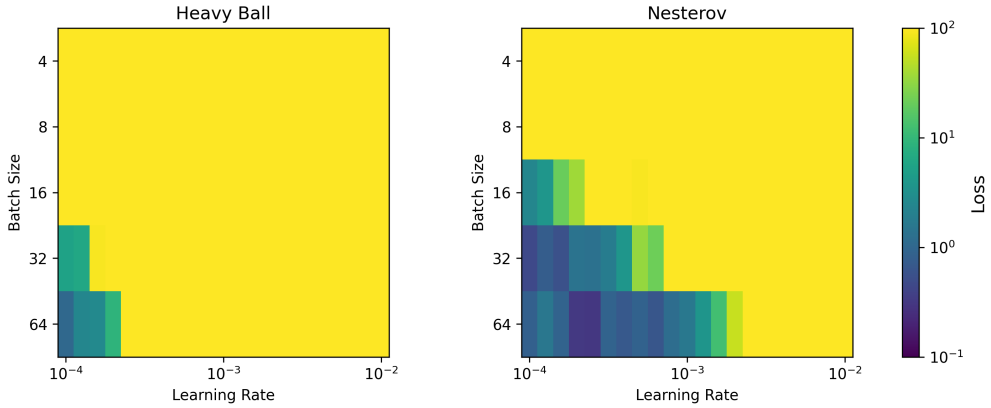
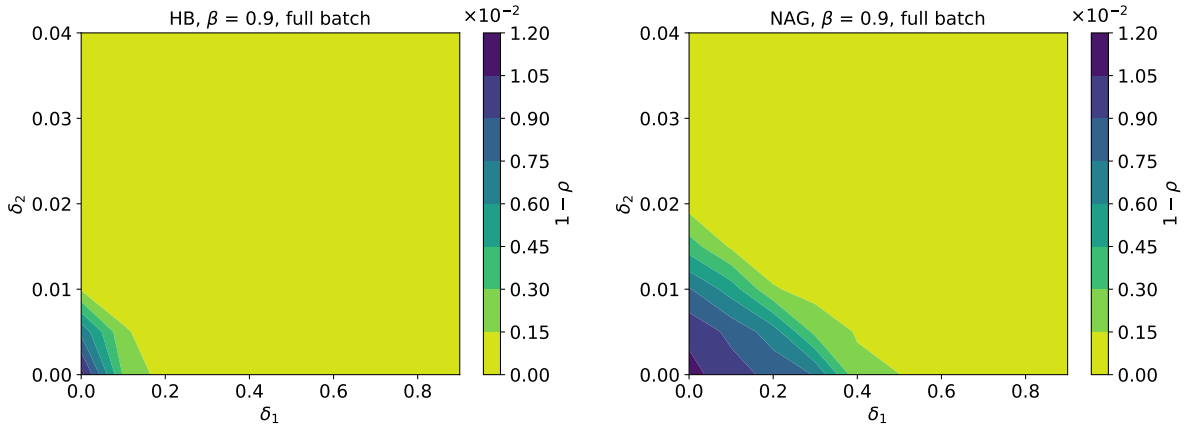
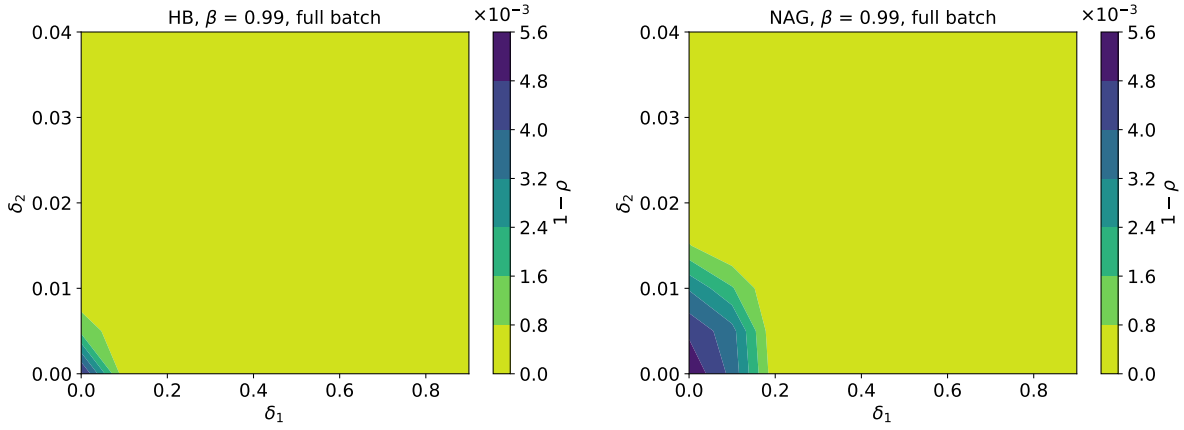


Figure 5: Training loss of a linear model on MNIST dataset with mean squared error loss, for heavy-ball and NAG with $\beta = 0.999$, with different batch sizes and learning rates. The loss is capped at 100. Large loss values indicate divergence. Compared to heavy-ball momentum, NAG is stable at smaller batch sizes and larger learning rates.



(a) Heavy-ball, $\beta = 0.9$

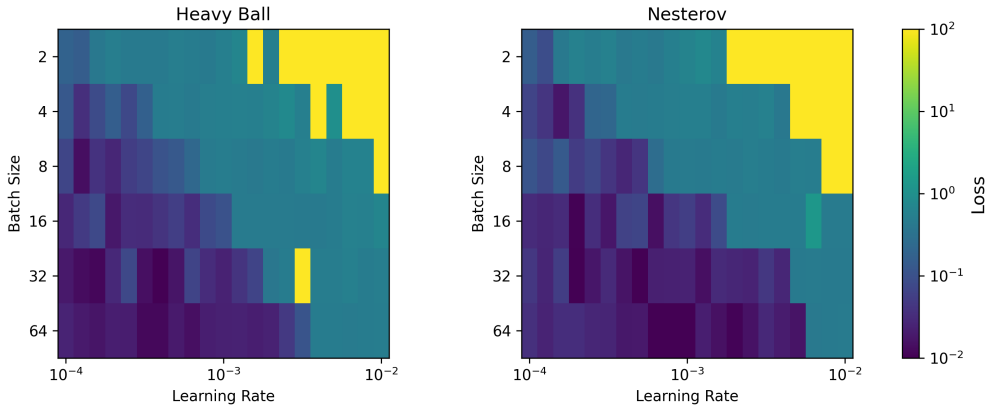
(b) Nesterov, $\beta = 0.9$



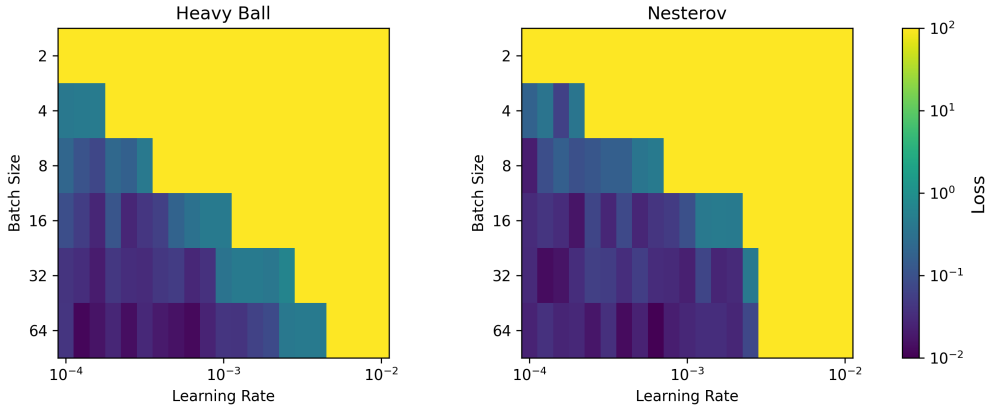
(c) Heavy-ball, $\beta = 0.99$

(d) Nesterov, $\beta = 0.99$

Figure 6: **NAG is more robust than HB when subjected to nonlinearity.** Convergence rate as a function of the amount of Jacobian uncertainty (full batch). For each point on the heatmap, we plot the best convergence rate searched over learning rate.



(a) Standard parameterization (SP)



(b) Maximal update parameterization (μP)

Figure 7: Training loss of an MLP model on MNIST dataset with mean squared error loss, for heavy-ball and NAG with $\beta = 0.999$, with different batch sizes and learning rates. The loss is capped at 100. Large loss values indicate divergence. The top and the bottom rows show the standard parameterization (SP) and the Maximal Update Parameterization (μP) (Yang et al., 2022), respectively. The training dynamics for μP is more nonlinear than SP. Both heavy-ball momentum and NAG require smaller batch sizes and learning rates to remain stable in μP compared to SP. But compared to heavy-ball momentum, NAG can tolerate smaller batch sizes and larger learning rates.

5 Related work on the comparison of HB and NAG

Theoretical & empirical results It is often observed in practice that NAG enjoys better performance and robustness than heavy-ball momentum (Sutskever et al., 2013; Choi et al., 2019). However, theoretical evidences are nuanced and often lacking. In the class of smooth, convex functions, NAG (with proper learning rate schedule) has a convergence rate $\mathcal{O}(1/k^2)$, which is optimal in first-order optimizers (Nesterov, 1983). In contrast, although heavy-ball momentum has local acceleration over gradient descent, it only has global linear convergence guarantee (with rate $\mathcal{O}(1/k)$) for smooth, convex functions, achieving no global acceleration (Ghadimi et al., 2015; Sun et al., 2019; Saab Jr et al., 2022; Shi et al., 2022). In more realistic scenarios, however, there is little theoretical evidence on the advantages of NAG. For certain types of overparameterized neural networks, NAG converges to the global minimum with a similar or worse rate than HB (Bu et al., 2021; Liu et al., 2022a;b). For stochastic setting with small batch sizes, neither momentum methods can outperform SGD (Kidambi et al., 2018).

"Lookahead" gradient in NAG There have been many works that attempt to explain NAG's superior performance and robustness. Intuitively, NAG evaluates the gradient at a partial update location, instead of at the current location as heavy-ball momentum (Figure 8). This "lookahead" step in gradient evaluation allows NAG to adapt the momentum in a quicker and more responsive way, and to reduce oscillation along high-curvature directions, which leads to NAG being more tolerant to larger momentum values (Sutskever et al., 2013). The analysis in Sutskever et al. (2013) gives helpful intuition, but lacks rigor. As we will show in the following paragraphs, the "lookahead" step has a more principled explanation using the control interpretations.

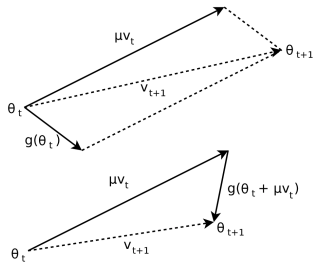


Figure 8: Taken from Figure 1 in Sutskever et al. (2013). (Top) Heavy-ball momentum. (Bottom) Nesterov accelerated gradient. Heavy-ball momentum evaluates the gradient $g(\theta_t)$ at the current location, while the NAG gradient evaluation $g(\theta_{t+1})$ has a "lookahead" step. In this plot from Sutskever et al. (2013), t is the time step (i.e. k in our notation), μ represents the momentum parameter (i.e. β in our notation), v represents the accumulated velocity vector that is eventually used to update the parameter θ.

Continuous-time limits and ODEs Another line of work focuses on the continuous-time limit of the optimizer dynamics. Polyak (1964) motivated the proposed momentum method as "the method of a small heavy sphere" that moves in a potential field (hence the name "heavy-ball momentum"). A more accurate physical analogy should be "a point mass moving in a potential field through a viscous medium" (Qian, 1999). The viscous medium causes friction that is proportional to the velocity. The dynamics can be described by the following second-order ODE:

$$m\ddot{x} + b\dot{x} + \nabla V(x) = 0, \tag{33}$$

where x is the position of the point mass (analogous to θ in optimizers), m is the mass, b is the viscous damping coefficient, and $V(x)$ is the potential field. We obtain the heavy-ball momentum method by discretizing Equation (33) using the forward Euler method (Qian, 1999). Let Δt be the time step, the discretized dynamics is:

$$m\frac{x_{t+\Delta t} - 2x_t + x_{t-\Delta t}}{\Delta t^2} + b\frac{x_{t+\Delta t} - x_t}{\Delta t} + \nabla V(x_t) = 0.$$

Rearranging, we recover the heavy-ball momentum method with learning rate $\eta = \frac{\Delta t^2}{m+b\Delta t}$ and momentum $\beta = \frac{m}{m+b\Delta t}$:

$$x_{t+\Delta t} = x_t + \frac{m}{m+b\Delta t}(x_t - x_{t-\Delta t}) - \frac{\Delta t^2}{m+b\Delta t}\nabla V(x_t).$$

However, the naive approach for taking continuous-time limit is not useful for distinguishing NAG from HB. If we take $\Delta t \to 0$, HB and NAG have the same limiting ODE (Equation (33)) (Shi et al., 2022). To address this, Shi et al. (2022) proposed an alternative limiting process that yields high-resolution ODEs, which differentiates NAG and HB. The high-resolution ODE for HB and NAG are given by:

$$\textbf{HB:}\quad m\ddot{x} + b\dot{x} + (1 + \frac{b\Delta t}{2m})\nabla V(x) = 0, \tag{34a}$$

$$\textbf{NAG:}\quad m\ddot{x} + \big(b + \frac{\Delta t}{m}\nabla^2 V(x)\big)\dot{x} + (1 + \frac{b\Delta t}{2m})\nabla V(x) = 0. \tag{34b}$$

The high-resolution ODEs have kept the Δt terms, and in the infinitesimal limit recover Equation (33). Notably, NAG has an additional damping term, which is proportional to the Hessian of the loss function. According to the ODE, NAG takes more cautious steps along high-curvature directions, which generally helps reduce oscillations. The high-resolution ODEs are shown to better match simulations of HB and NAG than the naive version, and empirical results support that NAG significantly reduces oscillation (Shi et al., 2022).

Control interpretations Rather than treating the optimizer as an inherent dynamical system that responds to an external force $\nabla V(x)$ (as in Equation (33)), we can adopt the opposite view and treat the optimizer as a controller (Hu & Lessard, 2017a). In the control view, $\nabla V(x)$ (analogous to gradient $\nabla\mathcal{L}$ in optimization) is the system to be controlled that has its own dynamics, and we would like to design a controller (optimizer) that stabilizes the system (make the gradient $\nabla\mathcal{L}$ go to zero). This is visualized in the block diagram in Figure 9. For readers unfamiliar with control systems concepts, we refer to Åström & Murray (2021) for a comprehensive introduction to feedback control theory.

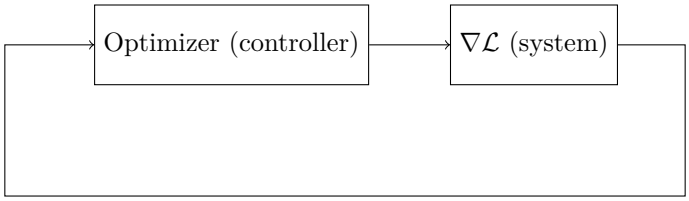


Figure 9: Block diagram showing the control view of the optimization process.

Since standard first-order optimizers[5] are linear time-invariant (LTI) systems, we can analyze their properties using transfer functions. Applying the z-transform to the optimizer update equations (eq. (5)), we get the following transfer functions for gradient descent, HB and NAG:

$$G_{\text{GD}}(z) = -\frac{\eta}{z-1}, \quad G_{\text{HB}}(z) = -\frac{\eta}{z-1}\cdot\frac{z}{z-\beta}, \quad G_{\text{NAG}}(z) = -\frac{\eta}{z-1}\cdot\frac{(1+\beta)z-\beta}{z-\beta}$$

From the transfer function, we can see that gradient descent corresponds to an integral controller. An integral controller (transfer function $\frac{K_I}{z-1}$, where K_I is the integral gain) produces control signal that is proportional to the sum of past errors, which is necessary to eventually drive the error to zero. Unsurprisingly, the transfer functions of HB and NAG both contain an integral control component.

The acceleration of the momentum methods is reflected by the additional components in their transfer functions. Heavy-ball has an additional $\frac{z}{z-\beta}$ term, which is a lag compensator. A lag compensator boosts

[5]By standard first-order optimizers, we refer to those with constant hyperparameters such as learning rate and momentum.

low-frequency gain and introduces a phase lag (Figure 10). In optimization, our objective is to asymptotically drive the gradient $\nabla\mathcal{L}$ to zero, which is a low frequency signal to track (in fact, frequency $\omega = 0$). A boost in low-frequency gain results in faster asymptotic convergence rate. This is apparent when we set the frequency $\omega = 0$, we have $z = e^{j\omega} = 1$, and $\frac{z}{z-\beta} = \frac{1}{1-\beta}$, recovering heavy-ball momentum's $\frac{1}{1-\beta}$ improvement in convergence rate for quadratic loss functions.

NAG is also the combination of an integrator and a lag compensator. With the same learning rate and momentum parameter, it has the same low-frequency gain as HB ($G_{NAG}(z) = G_{HB}(z)$ when $z = 1$), hence the same asymptotic convergence rate. However, it has a zero at $\frac{\beta}{1+\beta}$ instead of 0, which leads to a smaller phase lag. On the Bode plot (Figure 10), this leads to a flatter slope at the crossover frequency and a larger phase margin compared to HB, which contribute to NAG's improved robustness. The comparative fragility of HB may explain its lack of convergence guarantee for general strongly convex functions (Hu & Lessard, 2017a).

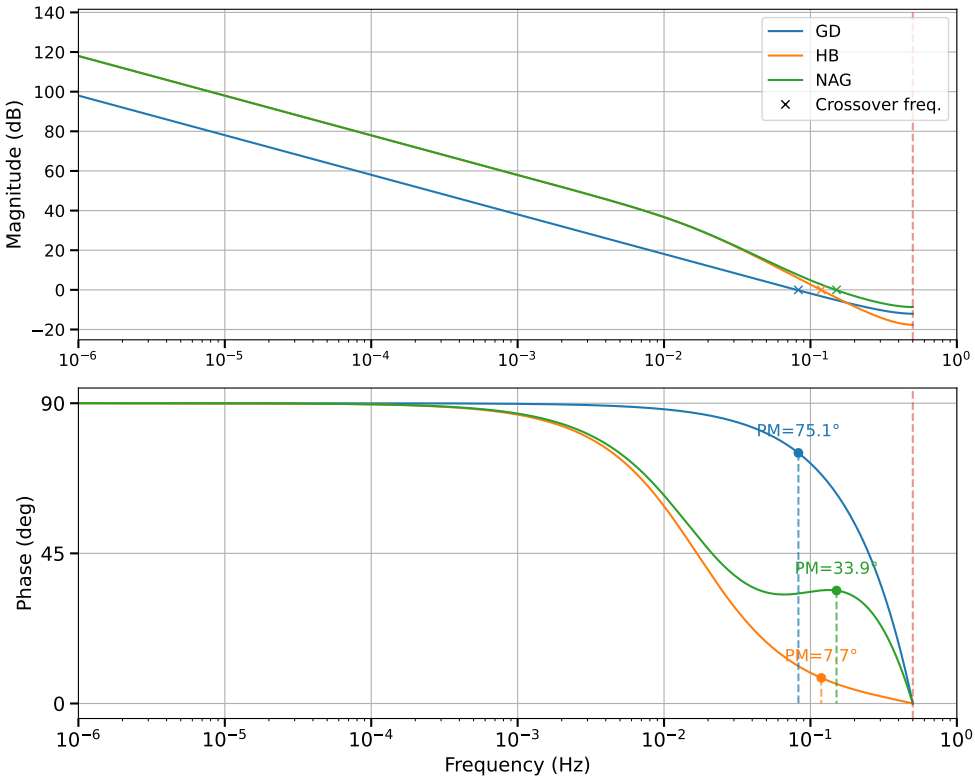


Figure 10: Bode plot showing gradient descent, HB and NAG (learning rate $\eta = 0.5$, momentum $\beta = 0.9$). The Bode plot shows the magnitude and phase of the optimizer responses, plotted against gradient signal frequency from 0 to the Nyquist frequency (with sampling rate 1, Nyquist freq. $= 1/2$, shown as the red dashed line). At low frequency, both HB and NAG have the same magnitude ($20\log_{10}\frac{1}{1-\beta} = 20$dB higher than GD), meaning that they both have $\frac{1}{1-\beta}$ improvement in asymptotic convergence rate for quadratic loss functions. However, the high frequency responses are different: NAG has a flatter slope at crossover frequency, and the smaller phase lag of NAG leads to a larger phase margin (the phase at crossover frequency, taken difference with 0° due to positive feedback) than HB. These indicate that NAG is more robust to high-frequency gradient noise.

Additionally, NAG can be viewed as incorporating derivative control in its dynamics (Hu & Lessard, 2017a). This can be shown if we perform a transformation of variables and let $u_k := (1+\beta)\theta_k - \beta\theta_{k-1}$, and then

rewrite the NAG update as:

$$u_{k+1} = u_k + \beta(u_k - u_{k-1}) - \eta\nabla\mathcal{L}(u_k) - \eta\beta(\nabla\mathcal{L}(u_k) - \nabla\mathcal{L}(u_{k-1})). \tag{35}$$

The last term in Equation (35) is a form of derivative control. Derivative control predicts future behavior based on local trends. This matches NAG's "lookahead" interpretation in Sutskever et al. (2013). Altogether, NAG can be viewed as a form of the celebrated Proportional–integral–derivative (PID) controller (Åström, 1995; Hu & Lessard, 2017a).

6 Other related work

The noisy quadratic model (NQM) (Zhang et al., 2019) gives important insights on how the optimization performance scales with batch sizes, and how such scaling interacts with factors such as momentum, moving average and preconditioning. This work extends the insights of NQM by adopting proper minibatch noise models, and by incorporating nonlinear training dynamics. Also, NQM relies on having an analytical expression for the iterate throughout the training process, which would be difficult for optimizers such as Nesterov accelerated gradient. In contrast, our framework can accommodate all optimizers that are LTI systems, which correspond to all first-order optimizers with constant hyperparameters. However, a limitation of our approach is that we focus on asymptotic performance, whereas NQM predicts finite-step loss values.

Lessard et al. (2016) first proposed to use the robust control framework for the analysis and design of optimizers. In this framework, the optimization is viewed as a dynamical system, and the convergence is expressed as the feasibility problem of a semi-definite program (SDP). Our analysis is based on the same framework. In Lessard et al. (2016), much of the effort is focused on designing integral-quadratic constraints (IQCs) for classical problem types in optimization (such as quadratic, smooth convex). In contrast, we focus on the case of neural network training, and propose a model for nonlinear dynamics and minibatch noise. Several follow-up works take dissipativity theory approach (Hu & Lessard, 2017b; Lessard, 2022), which are mathematically equivalent to the original IQC framework (Lessard et al., 2016), but may result in smaller LMIs. Hu et al. (2021) adapts the framework to stochastic gradient descent. However, they seek the worst-case performance with respect to the gradient noise, which is too conservative for minibatch training. Instead, we take the average-case performance with respect to our minibatch noise model.

A competing line of work is on the Performance Estimation Problem (PEP) (Drori & Teboulle, 2014; Taylor et al., 2017; 2018; Taylor & Bach, 2019). PEP also studies the worst-case performance of optimization problems by numerically solving SDPs. In contrast to the robust control based methods, PEP treats the optimization steps explicitly. This gives PEP the advantage of allowing finite-step loss bounds, whereas the robust control framework only certifies asymptotic performance. However, PEP scales poorly with the number of optimization steps, especially with complex problem classes and stochastic settings. We defer it to future work to obtain finite-step, NQM-like performance bounds using PEP.

7 Author contributions

The work was done in collaboration with John W. Simpson-Porco and Roger Grosse. XB derived the minibatch model, proved the theoretical results, and ran the simulations and experiments. RG suggested the multiplicative model for Jacobian uncertainty. JWSP helped deriving the convergence result with minibatch noise (Theorems 2 and 3). XB led the writing, with JWSP and RG providing detailed feedback. JWSP provided guidance regarding the robust control framework. RG provided general guidance throughout the project, especially regarding the contribution and relevance to machine learning optimization.

References

MOSEK ApS. *MOSEK Optimizer API for Python*, 2025. URL `https://docs.mosek.com/latest/pythonapi/index.html`.

Karl J Åström. Pid controllers: theory, design, and tuning. *The international society of measurement and control*, 1995.

Karl Johan Åström and Richard Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

Raghu Bollapragada, Tyler Chen, and Rachel Ward. On the fast convergence of minibatch heavy ball momentum. *IMA Journal of Numerical Analysis*, 45(3):1397–1424, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zhiqi Bu, Shiyun Xu, and Kan Chen. A dynamical view on optimization algorithms of overparameterized neural networks. In *International conference on artificial intelligence and statistics*, pp. 3187–3195. PMLR, 2021.

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 32, 2019.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization: a novel approach. *Mathematical Programming*, 145(1):451–482, 2014.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020.

Euhanna Ghadimi, Hamid Reza Feyzmahdavian, and Mikael Johansson. Global convergence of the heavy-ball method for convex optimization. In *2015 European control conference (ECC)*, pp. 310–315. IEEE, 2015.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Bin Hu and Laurent Lessard. Control interpretations for first-order optimization methods. In *2017 American Control Conference (ACC)*, pp. 3114–3119. IEEE, 2017a.

Bin Hu and Laurent Lessard. Dissipativity theory for nesterov's accelerated method. In *International Conference on Machine Learning*, pp. 1549–1557. PMLR, 2017b.

Bin Hu, Peter Seiler, and Laurent Lessard. Analysis of biased stochastic gradient descent using sequential semidefinite programs. *Mathematical programming*, 187:383–408, 2021.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9. IEEE, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Kiwon Lee, Andrew Cheng, Elliot Paquette, and Courtney Paquette. Trajectory of mini-batch momentum: batch size saturation and convergence in high dimensions. *Advances in Neural Information Processing Systems*, 35:36944–36957, 2022.

Laurent Lessard. The analysis of optimization algorithms: A dissipativity approach. *IEEE Control Systems Magazine*, 42(3):58–72, 2022.

Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

Xin Liu, Zhisong Pan, and Wei Tao. Provable convergence of nesterov's accelerated gradient method for over-parameterized neural networks. *Knowledge-Based Systems*, 251:109277, 2022a.

Xin Liu, Wei Tao, and Zhisong Pan. A convergence analysis of nesterov's accelerated gradient method in training deep linear neural networks. *Information Sciences*, 612:898–925, 2022b.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Charles H Martin and Michael W Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *Journal of Machine Learning Research*, 22(165): 1–73, 2021.

Takashi Mori, Liu Ziyin, Kangqiao Liu, and Masahito Ueda. Power-law escape rate of sgd. In *International Conference on Machine Learning*, pp. 15959–15975. PMLR, 2022.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, pp. 543, 1983.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

Samer Saab Jr, Shashi Phoha, Minghui Zhu, and Asok Ray. An adaptive polyak heavy-ball method. *Machine Learning*, 111(9):3245–3277, 2022.

Bin Shi, Simon S Du, Michael I Jordan, and Weijie J Su. Understanding the acceleration phenomenon via high-resolution differential equations. *Mathematical Programming*, pp. 1–70, 2022.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Samuel L Smith, Benoit Dherin, David Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021.

Tao Sun, Penghang Yin, Dongsheng Li, Chun Huang, Lei Guan, and Hao Jiang. Non-ergodic convergence analysis of heavy-ball algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5033–5040, 2019.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147. PMLR, 2013.

Adrien Taylor and Francis Bach. Stochastic first-order methods: non-asymptotic and computer-aided analyses via potential functions. In *Conference on Learning Theory*, pp. 2934–2992. PMLR, 2019.

Adrien Taylor, Bryan Van Scoy, and Laurent Lessard. Lyapunov functions for first-order methods: Tight automated convergence guarantees. In *International Conference on Machine Learning*, pp. 4897–4906. PMLR, 2018.

Adrien B Taylor, Julien M Hendrickx, and François Glineur. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161:307–345, 2017.

Nikhil Vyas, Yamini Bansal, and Preetum Nakkiran. Limitations of the ntk for understanding generalization in deep learning. *arXiv preprint arXiv:2206.10012*, 2022.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in neural information processing systems*, 32, 2019.

Liu Ziyin, Kangqiao Liu, Takashi Mori, and Masahito Ueda. Strength of minibatch noise in sgd. In *International Conference on Learning Representations*.

Liu Ziyin, Kangqiao Liu, Takashi Mori, and Masahito Ueda. Strength of minibatch noise in sgd. In *International Conference on Learning Representations*, 2022.

A Table of notations

Table 1 lists the important notations used in Sections 2 to 4.

Table 1: Important notations in Sections 2 to 4.

Symbol	Description
d	Number of model parameters
m	Dimension of the input vector
N	Number of datapoints in the dataset
θ	Model parameters
f_θ	Machine learning model, parameterized by θ
$\mathcal{X}$	Input space, $\mathcal{X} \subseteq \mathbb{R}^m$
$\mathcal{Y}$	Output space, $\mathcal{Y} \subseteq \mathbb{R}$
x	Input, $x \in \mathcal{X}$
y	Model output, $y = f_\theta(x) \in \mathcal{Y}$
t	Target, $t \in \mathcal{Y}$
$\boldsymbol{x}$	Vectorized input over a dataset
$\boldsymbol{f}_\theta$	Model with vectorized input $\boldsymbol{x}$, parameterized by θ
$\boldsymbol{y}$	Vectorized model output over a dataset, $\boldsymbol{y} = \boldsymbol{f}_\theta(\boldsymbol{x})$
$\boldsymbol{t}$	Vectorized target over a dataset
ℓ	Loss function, measuring the discrepancy of a model output and the corresponding target
$\mathcal{L}(\boldsymbol{x}, \boldsymbol{t}; \theta)$	Empirical risk (averaged loss over the dataset) as a function of input, target and parameters
k	Index indicating the optimization step
η	Learning rate
β	Momentum coefficient for heavy-ball momentum and Nesterov accelerated gradient
$L(\boldsymbol{y}, \boldsymbol{t})$	Empirical risk as a function of the output and target
g	Gradient of the empirical risk with respect to θ, i.e. $\nabla_\theta \mathcal{L}(\boldsymbol{x}, \boldsymbol{t}; \theta)$
ω	NQM gradient noise, i.e. i.i.d. Gaussian noise with covariance Σ
ϵ	i.i.d. spherical Gaussian noise
$\mathcal{B}$	Batch size
H	Hessian of the empirical risk with respect to θ, i.e. $\nabla^2_\theta \mathcal{L}(\boldsymbol{x}, \boldsymbol{t}; \theta)$
J	Jacobian, as the matrix of partial derivatives of the model output w.r.t. θ, i.e. $\frac{\partial \boldsymbol{f}_\theta(\boldsymbol{x})}{\partial \theta}$
ξ	State vector, in the state-space representation of a system
n	Dimension of the state vector
$\bar{A}, \bar{B}, \bar{C}$	Matrices characterizing a first-order optimizer in a state-space representation
$b^{(j)}, p^{(j)}$	Bernoulli random variables for the j-th datapoint, for deriving minibatch gradient.
U	The matrix whose columns contain the left singular vectors of J
u_j	j-th column of U
$\mathtt{y}, \varepsilon, \mathtt{J}$	Projected quantities, respectively y, ϵ, J multiplied by $U^\top$ on the left
$\mathtt{A}, \mathtt{A}_j, \mathtt{B}_\omega$	Matrices charactering the combined system, defined in eq. (19a)
ρ	Convergence rate
σ_j^2	Variance of the minibatch noise for the j-th datapoint
M, N	Multiplicative perturbation matrices for the Jacobian
p, q	The input and output signals for the combined system upon which we apply the constraints
$\delta_M, \delta_N, \delta_1, \delta_2, \delta_3$	ℓ_2 norm of the Jacobian perturbation matrices

B Aside: models with multidimensional outputs

We have so far focused on models that have a single output. However, the analysis of this paper can be easily extended to models with multidimensional outputs. For a model with multidimensional output, the

loss function is defined as

$$\ell(x,t,\theta)=\frac{1}{2}(\underline{f}_\theta(x)-\underline{t})^\top(\underline{f}_\theta(x)-\underline{t}),$$

where we use $\underline{f}_\theta(x),\underline{t}\in\mathbb{R}^n$ to denote the n-dimensional model output and label, respectively. With vectorized input $\boldsymbol{x}=(x^{(1)},\dots,x^{(N)})$, the vectorized output $\underline{\boldsymbol{f}}_\theta(\boldsymbol{x})\in\mathbb{R}^{N\times n}$. The Jacobian of $\underline{\boldsymbol{f}}_\theta(x)$ with respect to θ is a $N\times n\times d$ tensor. In order for the analysis for single-output models to hold, we can simply flatten the data (N) and the output (n) dimensions into a single dimension, i.e. replace $\boldsymbol{y}_k$ in eq. (11) with $\texttt{vec}(\boldsymbol{y}_k)\in\mathbb{R}^{Nn}$, and replace the Jacobian $J\in\mathbb{R}^{N\times n\times d}$ with $\texttt{flatten}(J,\texttt{axis}=(0,1))\in\mathbb{R}^{Nn\times d}$. All the remaining analysis exactly follows.

C Deferred derivation: weight space projection of minibatch gradient

In this section, we show the deferred derivation for the weight space projection of minibatch gradient in Section 3.2. In particular, we want to express the $\tilde{\nabla}^{\mathcal{B}}_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ using weight-space projected quantity $\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$.

We made Assumption 1, i.e. $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ is in the column space of U. This is a reasonable assumption, as we are only interested in the component of $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ that has an effect on updating θ.

To see this, let the projection of $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ onto the orthogonal complement of $\mathrm{col}(U)$ be $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})_\perp$, its contribution to the weight-space gradient $\nabla_\theta\mathcal{L}(\boldsymbol{x},\boldsymbol{t};\theta_k)_\perp$ is

$$\nabla_\theta\mathcal{L}(\boldsymbol{x},\boldsymbol{t};\theta_k)_\perp=J^\top\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})_\perp=VSU^\top(I-UU^\top)\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})=0,$$

indicating that the assumption $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})\in\mathrm{col}(U)$ is reasonable.

With Assumption 1, we can replace $\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ with $UU^\top\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$ in eq. (16), and get

$$U^\top\nabla^{\mathcal{B}}_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})=\frac{N}{\mathcal{B}}\sum_{j=1}^N b^{(j)}U^\top\mathbb{1}_{jj}UU^\top\nabla_{\boldsymbol{y}}L(\boldsymbol{y}_k,\boldsymbol{t})=\frac{N}{\mathcal{B}}\sum_{j=1}^N b^{(j)}u_ju_j^\top\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t}),\tag{36}$$

where u_j is the j-th column of U. On the right hand side of eq. (36), we have our projected output-space gradient $U^\top\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t})$.

For convenience, we also shift the Bernoulli random variable $b^{(j)}$ such that it has zero mean. Define $p^{(j)}=\frac{\mathcal{B}}{N}-b^{(j)}$, we have

$$\begin{aligned}\nabla^{\mathcal{B}}_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t})&=\frac{N}{\mathcal{B}}\sum_{j=1}^N(\frac{\mathcal{B}}{N}-p^{(j)})u_ju_j^\top\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t}),\qquad p^{(j)}=\begin{cases}\frac{\mathcal{B}}{N}-1, & \text{w. prob }\frac{\mathcal{B}}{N}\\ \frac{\mathcal{B}}{N}, & \text{w. prob }1-\frac{\mathcal{B}}{N}\end{cases}\\&=\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t})-\frac{N}{\mathcal{B}}\sum_{j=1}^N p^{(j)}u_ju_j^\top\nabla_{\mathrm{y}}L(\boldsymbol{y}_k,\boldsymbol{t}).\end{aligned}$$

D Deferred proofs

Proof of Theorem 1. Let ξ^* be the stationary point of the dynamical system described in eq. (9a). We have $\xi^*=0$, as we assumed without loss of generality that the NQM loss function is $\mathcal{L}_{\mathrm{NQM}}(\theta_k)=\frac{1}{2}\theta_k^\top H\theta_k$ has no offset term. However, in the derivation below, we keep the ξ^* terms for generality.

Multiply both sides of equation 10 by $\xi_l-\xi^*$, we get:

$$(\xi_l-\xi^*)^\top A^\top PA(\xi_l-\xi^*)-\rho^2(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)\le 0\tag{37}$$

Applying the dynamical equation equation 9a to $(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)$ and taking expectation, we get:

$$\begin{aligned}\mathbb{E}[(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)]&=\mathbb{E}_{\epsilon_{l-1}}[\big(A(\xi_{l-1}-\xi^*)+B_w\epsilon_{l-1}\big)^\top P\big(A(\xi_{l-1}-\xi^*)+B_w\epsilon_{l-1}\big)]\\&=\mathbb{E}[(\xi_{l-1}-\xi^*)^\top A^\top PA(\xi_{l-1}-\xi^*)]+2\mathbb{E}[(\xi_{l-1}-\xi^*)^\top A^\top PB_w\epsilon_{l-1}]+\mathbb{E}[\epsilon_{l-1}^\top B_w^\top PB_w\epsilon_{l-1}]\end{aligned}$$

The middle term reduces to zero because ϵ_{l-1} is a random variable with mean zero and is independent of the the state ξ_{l-1}. We have

$$\mathbb{E}[(\xi_l - \xi^*)^\top P(\xi_l - \xi^*)] = \mathbb{E}[(\xi_{l-1} - \xi^*)^\top A^\top PA(\xi_{l-1} - \xi^*)] + \frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w). \tag{38}$$

Taking the expectation of eq. (37), multiplying by ρ^{-2l} for each $l = 0, \ldots, k-1$ and summing over l, we get a telescoping series

$$\begin{aligned}
0 &\geq \sum_{l=0}^{k-1} \rho^{-2l}\mathbb{E}[(\xi_l - \xi^*)^\top A^\top PA(\xi_l - \xi^*) - \rho^2(\xi_l - \xi^*)^\top P(\xi_l - \xi^*)] \\
&= \rho^{-2(k-1)}\mathbb{E}[(\xi_{k-1} - \xi^*)^\top A^\top PA(\xi_{k-1} - \xi^*)] \\
&\quad - \rho^{-2(k-2)}\mathbb{E}[(\xi_{k-1} - \xi^*)^\top P(\xi_{k-1} - \xi^*)] + \rho^{-2(k-2)}\mathbb{E}[(\xi_{k-2} - \xi^*)^\top A^\top PA(\xi_{k-2} - \xi^*)] \\
&\quad - \cdots \\
&\quad - \rho^0\mathbb{E}[(\xi_1 - \xi^*)^\top P(\xi_1 - \xi^*)] + \rho^0\mathbb{E}[(\xi_0 - \xi^*)^\top A^\top PA(\xi_0 - \xi^*)] \\
&\quad - \rho^2\mathbb{E}[(\xi_0 - \xi^*)^\top P(\xi_0 - \xi^*)].
\end{aligned}$$

Substituting in eq. (38), we have

$$\begin{aligned}
0 \geq\ & \rho^{-2(k-1)}\left(\mathbb{E}[(\xi_k - \xi^*)^\top P(\xi_k - \xi^*)] - \frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w)\right) \\
&+ \rho^{-2(k-2)}\Big(- \mathbb{E}[(\xi_{k-1} - \xi^*)^\top P(\xi_{k-1} - \xi^*)] + \mathbb{E}[(\xi_{k-1} - \xi^*)^\top P(\xi_{k-1} - \xi^*)] \\
&\qquad\qquad - \frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w)\Big) \\
&+ \cdots \\
&+ \rho^0\Big(- \mathbb{E}[(\xi_1 - \xi^*)^\top P(\xi_1 - \xi^*)] + \mathbb{E}[(\xi_1 - \xi^*)^\top P(\xi_1 - \xi^*)] - \frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w)\Big) \\
&- \rho^2\mathbb{E}[(\xi_0 - \xi^*)^\top P(\xi_0 - \xi^*)].
\end{aligned}$$

Notice that many terms in the middle cancel, leaving us with the resulting inequality

$$\begin{aligned}
\rho^{-2(k-1)}\mathbb{E}[(\xi_k - \xi^*)^\top P(\xi_k - \xi^*)] - \rho^2\mathbb{E}[(\xi_0 - \xi^*)^\top P(\xi_0 - \xi^*)] \\
- \left(1 + \rho^{-2} + \cdots + \rho^{-2(k-1)}\right)\frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w) \leq 0.
\end{aligned}$$

Rearranging, we have:

$$\begin{aligned}
\mathbb{E}[(\xi_k - \xi^*)^\top P(\xi_k - \xi^*)] &\leq \rho^{2k}\mathbb{E}[(\xi_0 - \xi^*)^\top P(\xi_0 - \xi^*)] \\
&\quad + \rho^{2(k-1)}\left(1 + \rho^{-2} + \cdots + \rho^{-2(k-1)}\right)\frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w) &\text{(39a)} \\
&= \rho^{2k}\mathbb{E}[(\xi_0 - \xi^*)^\top P(\xi_0 - \xi^*)] + \frac{1-\rho^{2k}}{1-\rho^2}\cdot\frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w) &\text{(39b)}
\end{aligned}$$

We rewrite the NQM loss function $\mathcal{L}_{\mathrm{NQM}}(\theta_k) = \frac{1}{2}\theta_k^\top H\theta_k$ as a function of the state variable ξ_k, by substituting in eq. (9b). We have added the offset term $\xi^* = 0$ for completeness.

$$\mathcal{L}_{\mathrm{NQM}}(\xi_k) = \frac{1}{2}(\xi_k - \xi^*)^\top \bar{C}^\top H\bar{C}(\xi_k - \xi^*) = \frac{1}{2}(\xi_k - \xi^*)^\top M(\xi_k - \xi^*), \quad M := \bar{C}^\top H\bar{C}. \tag{40}$$

We would like to relate eq. (39) to the expected loss $\frac{1}{2}\mathbb{E}[(\xi_k - \xi^*)^\top M(\xi_k - \xi^*)]$. Since $H \succ 0$, we know that $M \succeq 0$. Also, because we are concerned with the loss, we can ignore the dimensions of M with zero

eigenvalues, as they do not contribute to the loss. Thus, we can assume $M \succ 0$. Therefore, we can replace P with $M^{\frac{1}{2}}M^{-\frac{1}{2}}PM^{-\frac{1}{2}}M^{\frac{1}{2}}$ in eq. (39).

Let $\sigma_{\min}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})$ and $\sigma_{\max}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})$ be the minimum and maximum eigenvalues of $M^{-\frac{1}{2}}PM^{-\frac{1}{2}}$, respectively. We can bound the left hand side of eq. (39b) as

$$\frac{1}{2}\mathbb{E}[(\xi_k-\xi^*)^\top M^{\frac{1}{2}}M^{-\frac{1}{2}}PM^{-\frac{1}{2}}M^{\frac{1}{2}}(\xi_k-\xi^*)] \geq \sigma_{\min}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})\cdot\frac{1}{2}\mathbb{E}[(\xi_k-\xi^*)^\top M(\xi_k-\xi^*)], \tag{41}$$

and the first term on the right hand side of eq. (39) as

$$\frac{1}{2}\mathbb{E}[(\xi_0-\xi^*)^\top M^{\frac{1}{2}}M^{-\frac{1}{2}}PM^{-\frac{1}{2}}M^{\frac{1}{2}}(\xi_0-\xi^*)] \leq \sigma_{\max}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})\cdot\frac{1}{2}\mathbb{E}[(\xi_0-\xi^*)^\top M(\xi_0-\xi^*)]. \tag{42}$$

Combining eqs. (39b), (41) and (42), we get

$$\begin{aligned}\frac{1}{2}\mathbb{E}[(\xi_k-\xi^*)^\top M(\xi_k-\xi^*)] \leq{}& \mathrm{cond}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})\cdot\rho^{2k}\frac{1}{2}\mathbb{E}[(\xi_0-\xi^*)^\top M(\xi_0-\xi^*)]\\ &+\sigma_{\min}^{-1}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})\cdot\frac{1-\rho^{2k}}{1-\rho^2}\cdot\frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w),\end{aligned}$$

where $\mathrm{cond}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})$ is the condition number of $M^{-\frac{1}{2}}PM^{-\frac{1}{2}}$. The expected loss $\mathbb{E}[\mathcal{L}_{\mathrm{NQM}}]$ has exponential convergence with rate ρ^2 to a steady-state risk upper bounded by $\sigma_{\min}^{-1}(M^{-\frac{1}{2}}PM^{-\frac{1}{2}})\cdot\frac{1}{1-\rho^2}\frac{1}{\mathcal{B}}\mathrm{Tr}(B_w^\top PB_w)$. □

Proof of Theorem 2. Similar to the proof of Theorem 1, we let ξ^* be the stationary point of the dynamical system described in eq. (20). We have $\xi^*=0$, but we keep the ξ^* terms in the derivation for generality. Multiply Equation (21) by $\xi_l-\xi^*$ on both sides, we have:

$$(\xi_l-\xi^*)^\top\mathtt{A}^\top P\mathtt{A}(\xi_l-\xi^*)-\rho^2(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)+\sum_{j=1}^N\sigma_j^2(\xi_l-\xi^*)^\top\mathtt{A}_j^\top P\mathtt{A}_j(\xi_l-\xi^*)\leq 0 \tag{43}$$

Applying the dynamical equation Equation (20) to $(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)$ and take expectation, we have:

$$\begin{aligned}\mathbb{E}[(\xi_l-\xi^*)^\top P(\xi_l-\xi^*)] = \mathbb{E}\Bigg[&\left(\mathtt{A}(\xi_{l-1}-\xi^*)+\sum_{j=1}^N\mathtt{A}_j(\xi_{l-1}-\xi^*)\cdot p_{l-1}^{(j)}+\mathtt{B}_\omega\varepsilon_{l-1}\right)^\top\\ &P\left(\mathtt{A}(\xi_{l-1}-\xi^*)+\sum_{j=1}^N\mathtt{A}_j(\xi_{l-1}-\xi^*)\cdot p_{l-1}^{(j)}+\mathtt{B}_\omega\varepsilon_{l-1}\right)\Bigg] &\text{(44a)}\\ =\mathbb{E}[(\xi_{l-1}-\xi^*)^\top\mathtt{A}^\top P\mathtt{A}(\xi_{l-1}-\xi^*)]+\sum_{j=1}^N\sigma_j^2\mathbb{E}[(\xi_{l-1}-\xi^*)^\top\mathtt{A}_j^\top P\mathtt{A}_j(\xi_{l-1}-\xi^*)]+\frac{1}{\mathcal{B}}\mathrm{Tr}(\mathtt{B}_\omega^\top P\mathtt{B}_\omega) &\text{(44b)}\\ =\mathbb{E}\left[(\xi_{l-1}-\xi^*)^\top\left(\mathtt{A}^\top P\mathtt{A}+\sum_{j=1}^N\sigma_j^2\mathtt{A}_j^\top P\mathtt{A}_j\right)(\xi_{l-1}-\xi^*)\right]+\frac{1}{\mathcal{B}}\mathrm{Tr}(\mathtt{B}_\omega^\top P\mathtt{B}_\omega) &\text{(44c)}\end{aligned}$$

Notice that the cross terms in eq. (44a) reduce to zero, as the random variables $p_{l-1}^{(j)}$ and ε_{l-1} have mean zero and are independent of the state ξ_{l-1} and of each other.

Taking the expectation of eq. (43), multiplying by ρ^{-2l} for each $l = 0, \dots, k-1$ and summing over l, we get a telescoping series

$$
\begin{aligned}
0 \geq \sum_{l=0}^{k-1} \rho^{-2l} \mathbb{E}[(\xi_l - \xi^*)^\top \mathtt{A}^\top P \mathtt{A} (\xi_l - \xi^*) - \rho^2 (\xi_l - \xi^*)^\top P (\xi_l - \xi^*) + \sum_{j=1}^{N} \sigma_j^2 (\xi_l - \xi^*)^\top \mathtt{A}_j^\top P \mathtt{A}_j (\xi_l - \xi^*)] \\
= \rho^{-2(k-1)} \mathbb{E}[(\xi_{k-1} - \xi^*)^\top \left(\mathtt{A}^\top P \mathtt{A} + \sum_{j=1}^{N} \sigma_j^2 \mathtt{A}_j^\top P \mathtt{A}_j \right) (\xi_{k-1} - \xi^*)] \\
\quad - \rho^{-2(k-2)} \mathbb{E}[(\xi_{k-1} - \xi^*)^\top P (\xi_{k-1} - \xi^*)] \\
\quad + \rho^{-2(k-2)} \mathbb{E}[(\xi_{k-2} - \xi^*)^\top \left(\mathtt{A}^\top P \mathtt{A} + \sum_{j=1}^{N} \sigma_j^2 \mathtt{A}_j^\top P \mathtt{A}_j \right) (\xi_{k-2} - \xi^*)] \\
\quad - \dots \\
\quad - \rho^0 \mathbb{E}[(\xi_1 - \xi^*)^\top P (\xi_1 - \xi^*)] + \rho^0 \mathbb{E}[(\xi_0 - \xi^*)^\top \left(\mathtt{A}^\top P \mathtt{A} + \sum_{j=1}^{N} \sigma_j^2 \mathtt{A}_j^\top P \mathtt{A}_j \right) (\xi_0 - \xi^*)] \\
\quad - \rho^2 \mathbb{E}[(\xi_0 - \xi^*)^\top P (\xi_0 - \xi^*)].
\end{aligned}
$$

Substituting in eq. (44c), we notice that many terms in the middle cancel, leaving us with

$$
\begin{aligned}
0 \geq -\rho^{-2(k-1)} \left(\mathbb{E}[(\xi_k - \xi^*)^\top P (\xi_k - \xi^*)] - \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega) \right) \\
\quad - \left(\rho^{-2(k-2)} + \dots + \rho^0 \right) \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega) \\
\quad - \rho^2 \mathbb{E}[(\xi_0 - \xi^*)^\top P (\xi_0 - \xi^*)] \\
= -\rho^{-2(k-1)} \mathbb{E}[(\xi_k - \xi^*)^\top P (\xi_k - \xi^*)] - \rho^2 \mathbb{E}[(\xi_0 - \xi^*)^\top P (\xi_0 - \xi^*)] \\
\quad - \left(1 + \rho^{-2} + \dots + \rho^{-2(k-1)} \right) \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega).
\end{aligned}
$$

We rewrite the expected loss $\mathbb{E}[\frac{1}{2} \mathtt{y}_k^\top \mathtt{y}_k]$ as a function of the state variable ξ_k, by substituting in eq. (20b) (we have added the offset term $\xi^* = 0$ for completeness):

$$
\frac{1}{2} \mathbb{E}[\mathtt{y}_k^\top \mathtt{y}_k] = \frac{1}{2} \mathbb{E}[(\xi_k - \xi^*)^\top \bar{C}^\top \mathtt{J}^\top \mathtt{J} \bar{C} (\xi_k - \xi^*)].
$$

The rest of the proof follows the same steps as in the proof of Theorem 1. □

Proof of Theorem 3. Let $V(\xi_k) = \xi_k^\top P \xi_k$ for some $P \succ 0$. Define $\mathrm{p}_k := \left[p_{1,k}^\top, p_{2,k}^\top, p_{3,k}^\top\right]^\top$. Using the dynamics in eq. (28), we have

$$\begin{aligned}
\mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[V(\xi_{k+1})] - \rho^2 V(\xi_k) = \mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[(\hat{A}\xi_k + \hat{B}\mathrm{p}_k + \sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)} + \mathtt{B}_\omega \varepsilon_k)^\top P \\
(\hat{A}\xi_k + \hat{B}\mathrm{p}_k + \sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)} + \mathtt{B}_\omega \varepsilon_k)] - \rho^2 \xi_k^\top P \xi_k \\
= \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
+ \mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[2(\hat{A}\xi_k + \hat{B}\mathrm{p}_k)^\top P\big(\sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)} + \mathtt{B}_\omega \varepsilon_k\big)] \\
+ \mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)}}[\big(\sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)}\big)^\top P\big(\sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)}\big)] \\
+ \mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[2\big(\sum_{j=1}^N \mathtt{A}_j \xi_k \cdot r_k^{(j)}\big)^\top P(\mathtt{B}_\omega \varepsilon_k)] \\
+ \mathbb{E}_{\varepsilon_k}[(\mathtt{B}_\omega \varepsilon_k)^\top P(\mathtt{B}_\omega \varepsilon_k)]
\end{aligned}$$

Because $p_k^{(1)}, \ldots, p_k^{(N)}, \varepsilon_k$ are independent random variables, and $\mathbb{E}[r^{(j)}] = 0, j = 1, \ldots, N$, $\mathbb{E}[\varepsilon_k] = 0$, the first order terms for $r_k^{(j)}$ and ε_k drop out. We have

$$\begin{aligned}
\mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[V(\xi_{k+1})] - \rho^2 V(\xi_k) &= \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
&\quad + \sum_{j=1}^N \mathbb{E}_{r_k^{(j)}}[(\mathtt{A}_j \xi_k \cdot r_k^{(j)})^\top P(\mathtt{A}_j \xi_k \cdot r_k^{(j)})] \\
&\quad + \mathbb{E}_{\varepsilon_k}[(\mathtt{B}_\omega \varepsilon_k)^\top P(\mathtt{B}_\omega \varepsilon_k)] \qquad (45a) \\
&= \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
&\quad + \sum_{j=1}^N \mathbb{E}[r_k^{(j)} r_k^{(j)}] \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix}^\top P \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
&\quad + \mathrm{Tr}(\mathbb{E}_{\varepsilon_k}[\varepsilon_k^\top \varepsilon_k] \mathtt{B}_\omega^\top P \mathtt{B}_\omega) \qquad (45b) \\
&= \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \Bigg(\begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} \\
&\qquad + \sum_{j=1}^N \sigma_j^2 \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix}^\top P \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix} \Bigg) \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} + \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega) \qquad (45c)
\end{aligned}$$

We would like to bound the first term in eq. (45c). To do that, we use the LMI (eq. (32)) and the condition in eq. (31). Multiply the matrices in eq. (32) by $\begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top$ and its transpose on both sides respectively, we have

$$\begin{aligned}
\begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \left(\begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} + \sum_{j=1}^{N} \sigma_j^2 \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix}^\top P \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix} \right) \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
+\lambda_1 \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix}\hat{C}_1 & \hat{D}_1 & 0 & 0 \\ 0 & I & 0 & 0\end{bmatrix}^\top \begin{bmatrix}\delta_1^2 I & 0 \\ 0 & -I\end{bmatrix} \begin{bmatrix}\hat{C}_1 & \hat{D}_1 & 0 & 0 \\ 0 & I & 0 & 0\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
+\lambda_2 \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix}\hat{C}_2 & 0 & \hat{D}_2 & 0 \\ 0 & 0 & I & 0\end{bmatrix}^\top \begin{bmatrix}\delta_2^2 I & 0 \\ 0 & -I\end{bmatrix} \begin{bmatrix}\hat{C}_2 & 0 & \hat{D}_2 & 0 \\ 0 & 0 & I & 0\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \\
+\lambda_3 \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \begin{bmatrix}\hat{C}_3 & 0 & 0 & \hat{D}_3 \\ 0 & 0 & 0 & I\end{bmatrix}^\top \begin{bmatrix}\delta_3^2 I & 0 \\ 0 & -I\end{bmatrix} \begin{bmatrix}\hat{C}_3 & 0 & 0 & \hat{D}_3 \\ 0 & 0 & 0 & I\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \leq 0. \qquad (46)
\end{aligned}$$

Notice that

$$\begin{aligned}
\begin{bmatrix}\hat{C}_1 & \hat{D}_1 & 0 & 0 \\ 0 & I & 0 & 0\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} = \begin{bmatrix}q_{1,k} \\ p_{1,k}\end{bmatrix} = \begin{bmatrix}q_{1,k} \\ \Delta_1 q_{1,k}\end{bmatrix}, \\
\begin{bmatrix}\hat{C}_2 & 0 & \hat{D}_2 & 0 \\ 0 & 0 & I & 0\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} = \begin{bmatrix}q_{2,k} \\ p_{2,k}\end{bmatrix} = \begin{bmatrix}q_{2,k} \\ \Delta_2 q_{2,k}\end{bmatrix}, \\
\begin{bmatrix}\hat{C}_3 & 0 & 0 & \hat{D}_3 \\ 0 & 0 & 0 & I\end{bmatrix} \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} = \begin{bmatrix}q_{3,k} \\ p_{3,k}\end{bmatrix} = \begin{bmatrix}q_{3,k} \\ \Delta_3 q_{3,k}\end{bmatrix}.
\end{aligned}$$

Using the conditions in eq. (31), we know that the last three terms on the left hand side of eq. (46) are non-negative. Therefore, eq. (46) reduces to

$$\begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix}^\top \left(\begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix}^\top \begin{bmatrix} -\rho^2 P & \\ & P\end{bmatrix} \begin{bmatrix} I & \\ \hat{A} & \hat{B}\end{bmatrix} + \sum_{j=1}^{N} \sigma_j^2 \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix}^\top P \begin{bmatrix}\mathtt{A}_j & 0\end{bmatrix} \right) \begin{bmatrix}\xi_k \\ \mathrm{p}_k\end{bmatrix} \leq 0.$$

Substituting into eq. (45), we have

$$\mathbb{E}_{p_k^{(1)},\ldots,p_k^{(N)},\varepsilon_k}[V(\xi_{k+1})] - \rho^2 V(\xi_k) \leq \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega).$$

We can then apply the same telescoping sum as in the proofs of Theorem 1 and Theorem 2 to show that for all $k \geq 0$,

$$0 \geq \rho^{-2(k-1)} \mathbb{E}[V(\xi_k)] - \rho^2 V(\xi_0) - \left(1 + \rho^{-2} + \cdots + \rho^{-2(k-1)}\right) \cdot \frac{1}{\mathcal{B}} \mathrm{Tr}(\mathtt{B}_\omega^\top P \mathtt{B}_\omega).$$

The rest of the proof follows the same steps as in the proof of Theorem 1 and Theorem 2. □

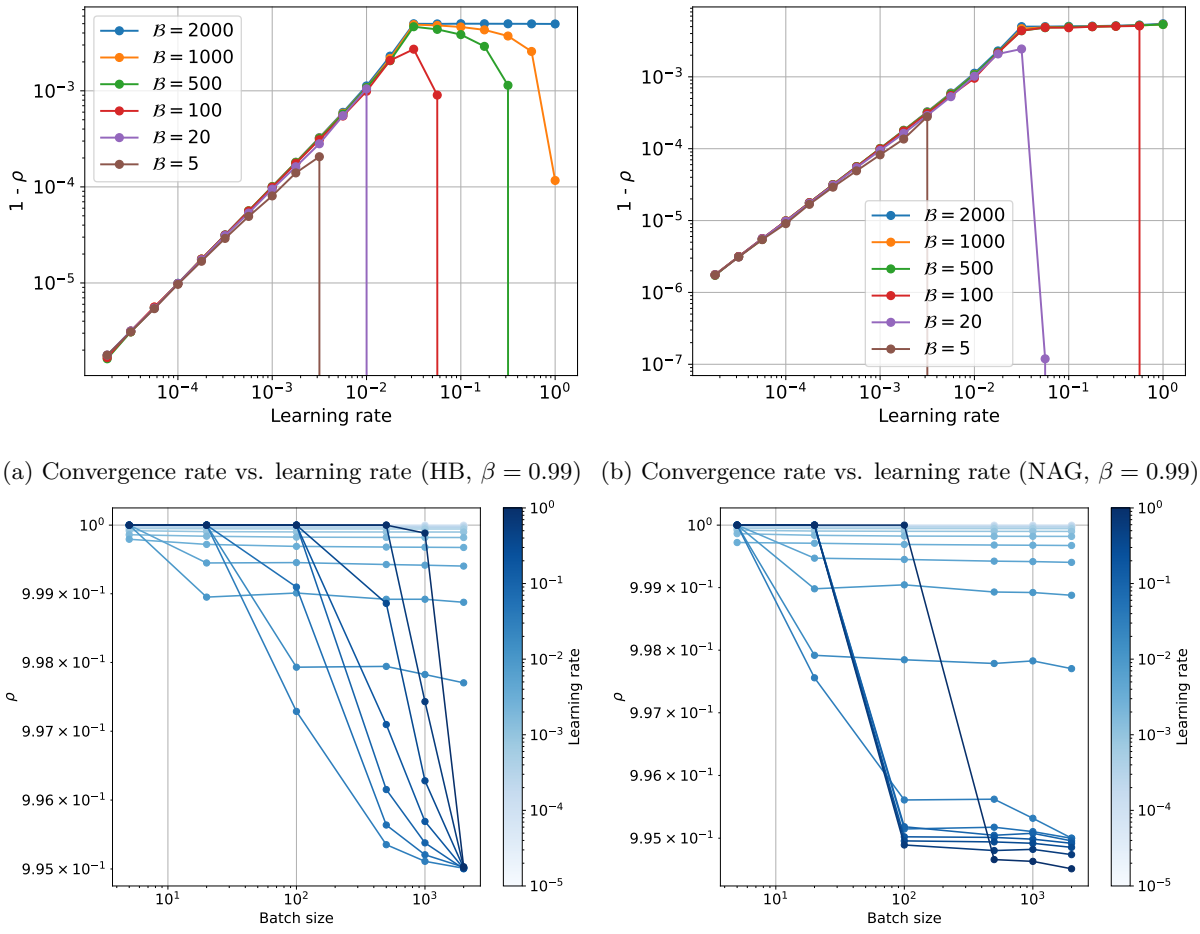
E Simulation details

Table 2 summarizes the MOSEK solver parameters used in the simulations. For the minibatch noise model, we randomly sample the u_j vectors in eq. (20) from the orthogonal group of dimension equal to the number of bins, independently for $j = 1, \ldots, N$.

F Additional simulation results

Table 2: MOSEK solver parameters. A detailed description of the parameters can be found in the MOSEK documentation (ApS, 2025).

Parameter name	Value
MSK_DPAR_BASIS_TOL_X	1e-9
MSK_DPAR_BASIS_TOL_S	1e-9
MSK_DPAR_BASIS_REL_TOL_S	1e-20
MSK_IPAR_INTPNT_MAX_ITERATIONS	1000000
MSK_DPAR_SEMIDEFINITE_TOL_APPROX	1e-15
MSK_DPAR_PRESOLVE_TOL_X	1e-12
MSK_DPAR_INTPNT_CO_TOL_DFEAS	1e-11
MSK_DPAR_INTPNT_CO_TOL_PFEAS	1e-13
MSK_DPAR_INTPNT_CO_TOL_REL_GAP	1e-13



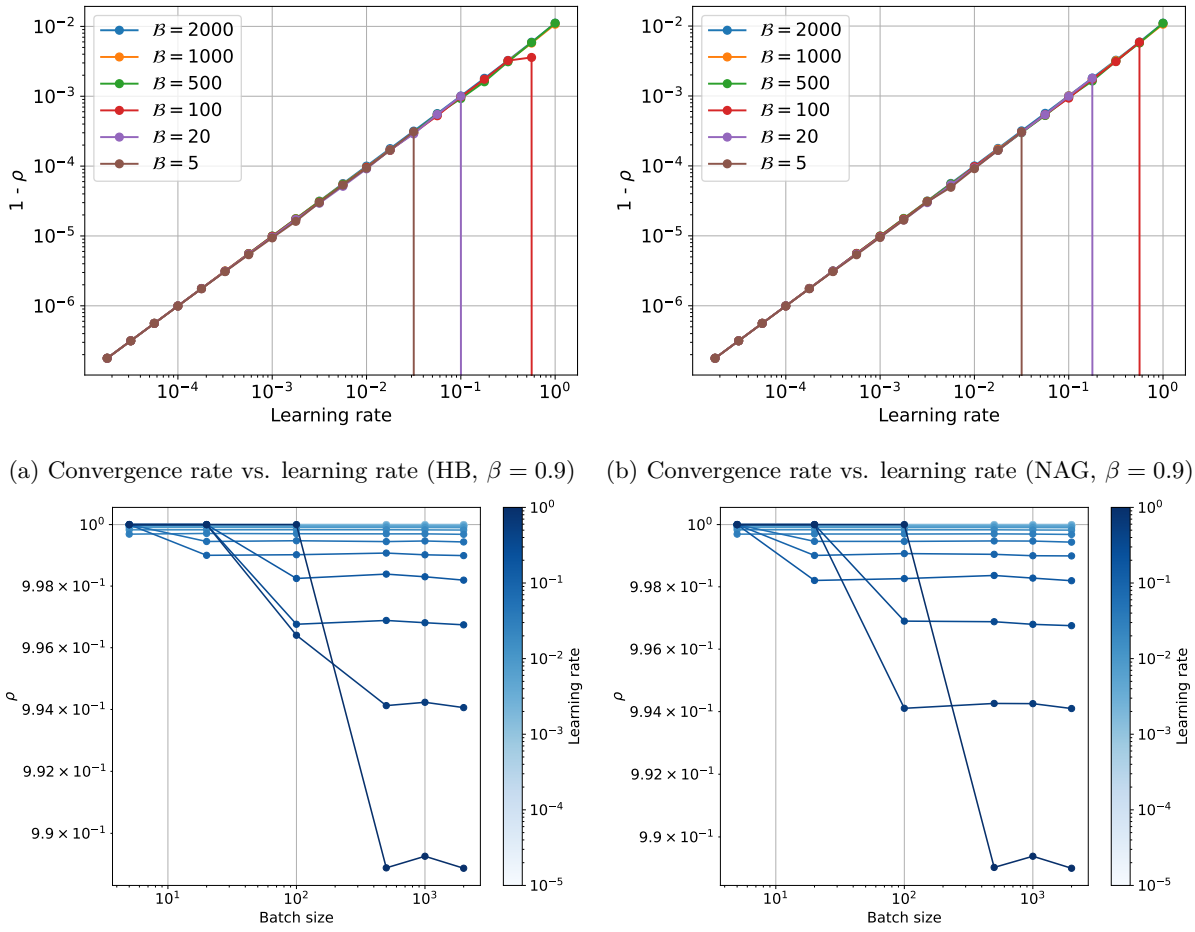
(a) Convergence rate vs. learning rate (HB, $\beta = 0.99$)

(b) Convergence rate vs. learning rate (NAG, $\beta = 0.99$)

(c) Convergence rate vs. batch size (HB, $\beta = 0.99$)

(d) Convergence rate vs. batch size (NAG, $\beta = 0.99$)

Figure 11: **With large momentum, NAG is more robust than HB at larger learning rates at small batch sizes.** With large momentum and learning rates, the system becomes overdamped.



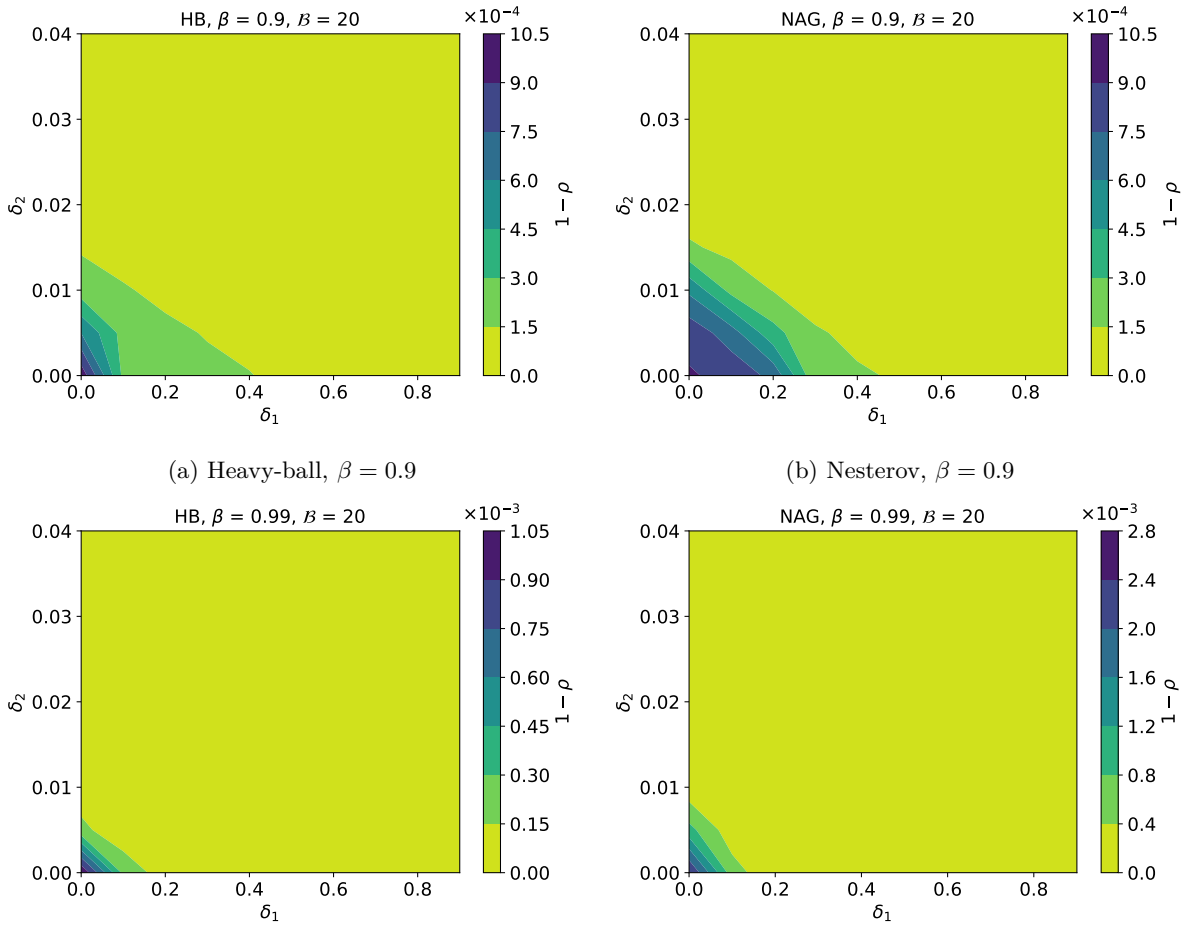
(a) Convergence rate vs. learning rate (HB, $\beta = 0.9$)

(b) Convergence rate vs. learning rate (NAG, $\beta = 0.9$)

(c) Convergence rate vs. batch size (HB, $\beta = 0.9$)

(d) Convergence rate vs. batch size (NAG, $\beta = 0.9$)

Figure 12: **With large momentum, NAG is more robust than HB at larger learning rates at small batch sizes.** With large momentum and learning rates, the system becomes overdamped.



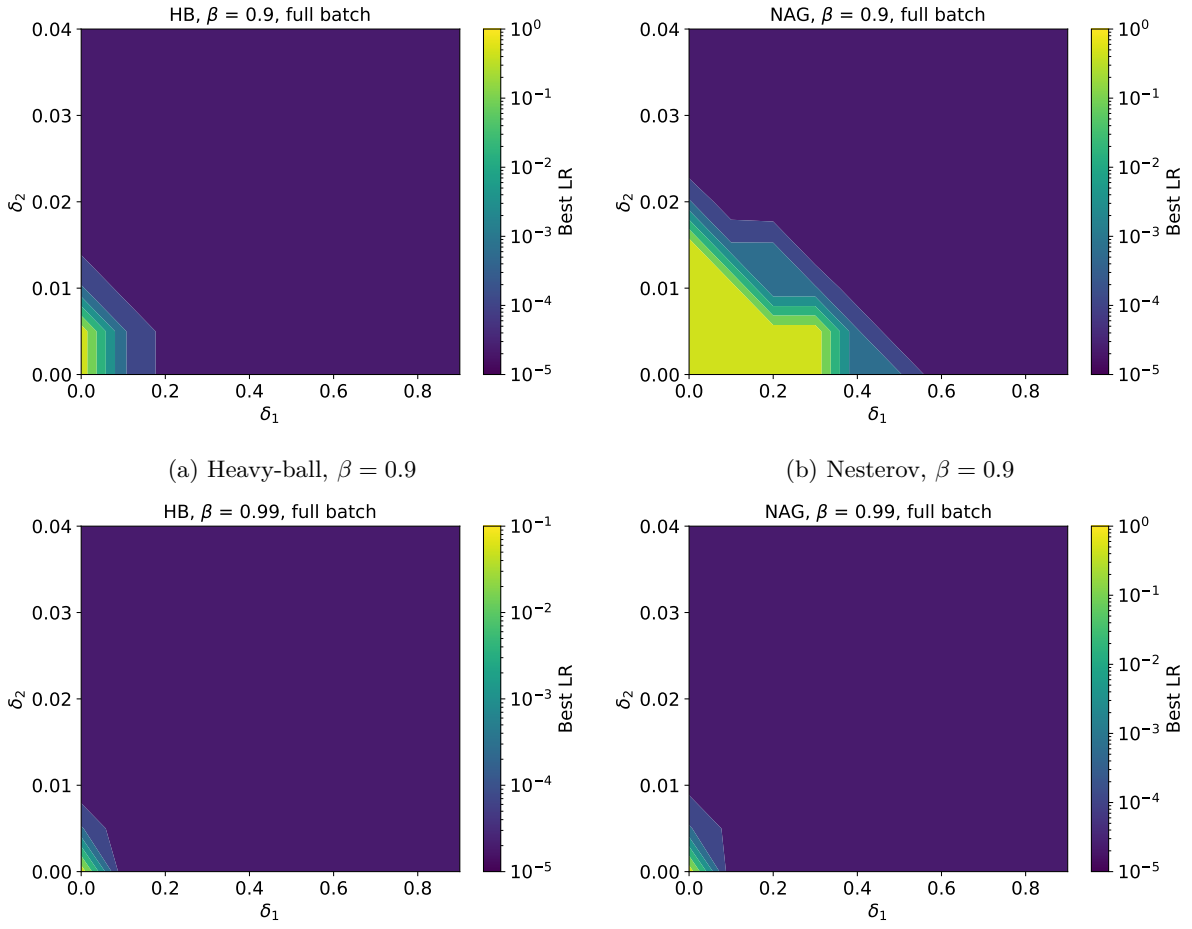
(a) Heavy-ball, $\beta = 0.9$

(b) Nesterov, $\beta = 0.9$

(c) Heavy-ball, $\beta = 0.99$

(d) Nesterov, $\beta = 0.99$

Figure 13: Convergence rate as a function of the amount of Jacobian uncertainty (batch size 20). For each point on the heatmap, we plot the best convergence rate searched over learning rate.



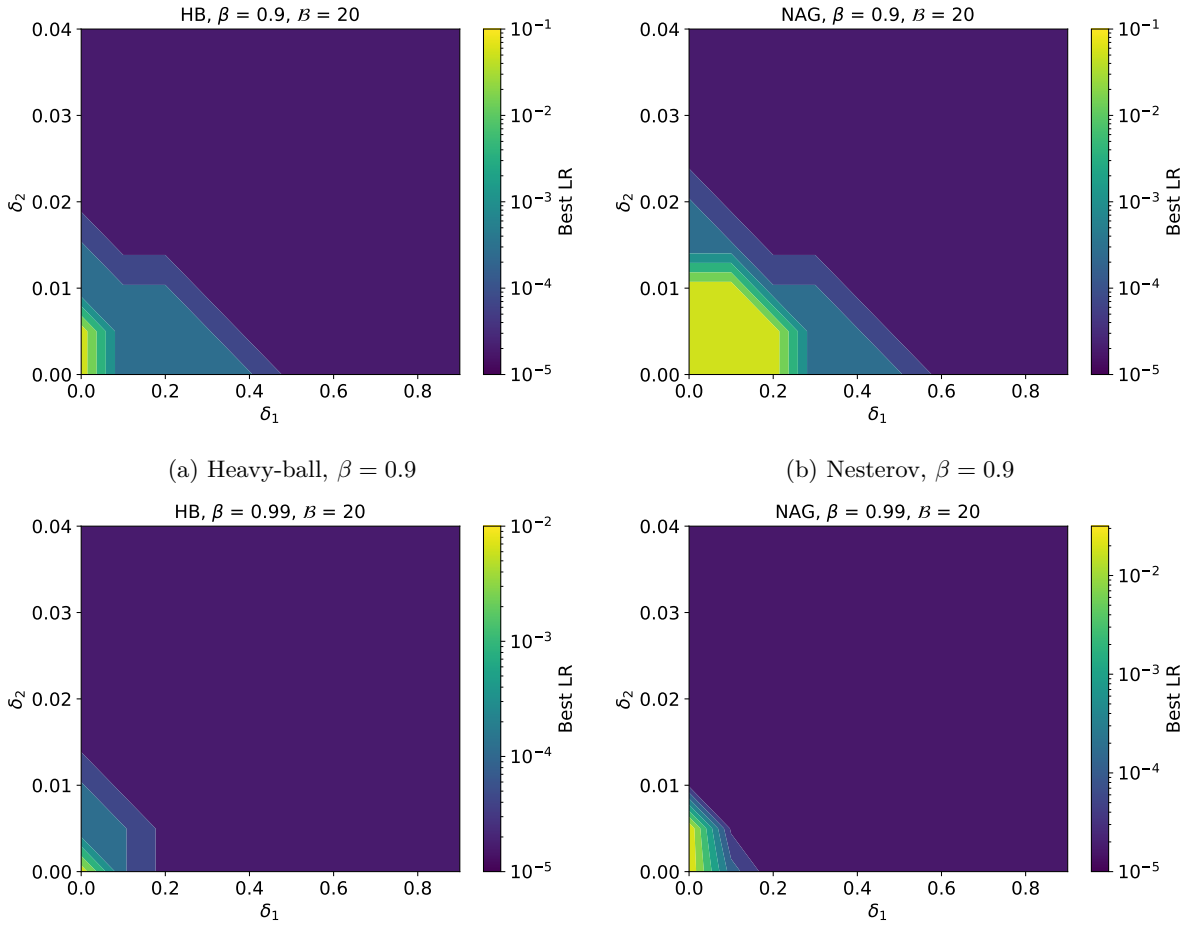
(a) Heavy-ball, $\beta = 0.9$

(b) Nesterov, $\beta = 0.9$

(c) Heavy-ball, $\beta = 0.99$

(d) Nesterov, $\beta = 0.99$

Figure 14: Optimal learning rate used to certify the convergence rates in Figure 6.



(a) Heavy-ball, $\beta = 0.9$

(b) Nesterov, $\beta = 0.9$

(c) Heavy-ball, $\beta = 0.99$

(d) Nesterov, $\beta = 0.99$

Figure 15: Optimal learning rate used to certify the convergence rates in Figure 13.