IEEE Copyright Notice

Cite as:

Tune smarter not harder: A principled approach to tuning learning rates for shallow nets

Thulasi Tholeti*   Sheetal Kalyani*

Abstract—**Effective hyper-parameter tuning is essential to guarantee the performance that neural networks have come to be known for. In this work, a principled approach to choosing the learning rate is proposed for shallow feedforward neural networks. We associate the learning rate with the gradient Lipschitz constant of the objective to be minimized while training. An upper bound on the mentioned constant is derived and a search algorithm, which always results in non-divergent traces, is proposed to exploit the derived bound. It is shown through simulations that the proposed search method significantly outperforms the existing tuning methods such as Tree Parzen Estimators (TPE). The proposed method is applied to three different existing applications: a) channel estimation in OFDM systems, b) prediction of the exchange currency rates and c) offset estimation in OFDM receivers, and it is shown to pick better learning rates than the existing methods using the same or lesser compute power.**

I. INTRODUCTION

Deep neural networks have made significant improvements to fields like speech and image processing [1], communications [2]–[4], computer vision, etc. [5]. These networks are typically trained using an iterative optimization algorithm such as Gradient Descent (GD) or its multiple variants [6], [7]. To successfully deploy these networks for various applications, the hyper-parameters of the network, namely the width and the depth of the network and the learning rate used for training should be carefully tuned [8].

Initially, manual search and grid search were the most popular approaches [9]. The authors of [10] then showed that randomly chosen trials were more efficient in terms of search time for hyper-parameter optimization than a grid-based search. However, in both the methods, the observations from the previous samples are not utilized to choose values for the subsequent trials. To remedy this, Sequential Model-Based Optimization (SMBO) was introduced to perform hyper-parameter tuning where the next set of hyper-parameters to be evaluated are chosen based on the previous trials [11]. Some of the well-known models for Bayesian optimization are Gaussian Processes [12], random forests [13] and TPE [14].

In the methods listed here so far, the tuning of hyper-parameters is typically performed as a black-box module, i.e., without utilizing any information about the objective function to be minimized. There exist many applications in which the architecture of the network is fixed, for which the number of layers and the width of the network are already specified

*The authors are with the Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai, India 600 036. Email: {ee15d410,skalyani}@ee.iitm.ac.in

and are not treated as hyper-parameters. Given such an architecture, the learning rate is an important hyper-parameter as it determines the speed of convergence of the optimization algorithm [15]. In such cases, it would be beneficial if the learning rate is derived as a function of the objective as it can be simply recomputed for a new set of inputs instead of tuning the learning rate from scratch.

The idea of tuning-free algorithms has recently attracted attention, not only in neural networks but in the context of other algorithms as well. For example, [16] proposed a tuning-free Orthogonal Matching Pursuit (OMP) algorithm, [17] proposed a tuning-free hedge algorithm and [18] proposed a parameter-free robust Principal Component Analysis (PCA) method. To propose such a tuning-free equivalent for the GD algorithm while training neural networks, it would require a theoretical analysis of the objective function. Although neural networks are applied to varied applications, little is known about its theoretical properties when the network consists of multiple hidden layers. Most theoretical works such as [19], [20] are available for networks with one or two hidden layers, which we call shallow networks.

Although deep neural networks are popular in computer vision and image processing where the objective function is complex, applications in areas like wireless communication and finance predictions still employ shallow feedforward neural networks as evidenced by works in [21]–[25]. In [22], channel estimation for Orthogonal Frequency Division Multiplexing (OFDM) systems was done using a single hidden layer network. Shallow networks were also used in applications like user equipment localization [25], symbol detection in high-speed OFDM underwater acoustic communication [26] and Direction of Arrival (DoA) estimation [24]. In all the above, the architecture for a given application was fixed and the learning rate was chosen by manual tuning or grid search.

For such applications which employ a fixed shallow architecture, a theory-based approach for choosing the learning rate will save the computation which would otherwise be spent on tuning hyper-parameters. The learning rate of the optimization algorithm has been associated with the Lipschitz properties of the objective function, namely the Lipschitz constant of the gradient of the objective function in [27]. Although, there has been significant interest in analyzing the Lipschitz properties of neural networks in recent literature [28], [29], these works focus on the Lipschitz constant of the output which plays an important role in analysing the stability of the network, and not on the gradient Lipschitz constant of the objective which is required for quantifying the learning rate.

A. Motivation

In the existing works on hyper-parameter optimization, the choice of learning rate is often treated as a separate module that is to be performed before the training; they do not employ any information about the function that should be optimized. As an alternative, we wish to associate the learning rate with the parameters of the problem, thereby providing a theoretical justification to the choice of learning rate and also use this to tune in a smarter fashion. In typical tuning methods, there is a clear trade-off between the number of trials of the search algorithm that is allowed and the performance of the chosen learning rate. If one decides to adopt a higher number of trials then, one is more likely to achieve a better learning rate. However, there is no guarantee that the chosen learning rate will lead to convergent behaviour of GD given any fixed number of trials. In the proposed method, we wish to provide the user with the same trade-off between the number of trials and the performance, whilst ensuring that chosen learning rate always results in convergence irrespective of the number of trials allowed.

B. Contributions

A theory-based approach to determine the learning rate for shallow networks is proposed. The contributions of this work are four-fold. Firstly, using classic literature [27], the learning rate is associated with the gradient Lipschitz constant of the objective function. Secondly, the upper bound on gradient Lipschitz constants for feedforward neural networks consisting of one and two layers are derived for popular activation functions, namely, ReLU and sigmoid. The bounds, initially in terms of eigenvalues of large Hessian matrices, are simplified to yield easy-to-implement expressions that can be adapted to a given architecture. Thirdly, the derived bound on the gradient Lipschitz constant is utilized for determining the learning rate; an algorithm, 'BinarySearch', is introduced for this search. The proposed algorithm is shown to outperform the popular hyper-parameter tuning estimator, TPE, in terms of the loss achieved, while ensuring convergence. Finally, the utility of the proposed method is also demonstrated using three applications: channel estimation in the case of OFDM systems, Carrier Frequency Offset estimation in OFDM receivers and the prediction of exchange rates for currencies.

C. Notation

We use bold upper-case letters, say $\boldsymbol{A}$ to denote matrices and A_{ij}, $\boldsymbol{A}^i$ to denote their $(i,j)_{th}$ element and the i_{th} column respectively. The maximum eigenvalue of $\boldsymbol{A}$ is denoted as $\lambda_{max}(\boldsymbol{A})$; the maximum diagonal entry is denoted as $\mathcal{D}_{max}(\boldsymbol{A})$. The bold lower-case letters $\boldsymbol{x}, \boldsymbol{y}$ denote vectors. All vectors are column vectors unless stated otherwise. The ℓ_2 norm of a vector is denoted as $\|.\|$. The ℓ_1 and ℓ_∞ norms of a vector $\boldsymbol{x}$ are denoted as $|\boldsymbol{x}|_1 = \sum_i x_i$ and $|\boldsymbol{x}|_\infty = \max_i x_i$ respectively. The indicator function denoted as $\mathbb{I}_\mathcal{E}$ takes the value 1 when $\mathcal{E}$ is true and value 0 otherwise. The symbols ∇ and ∇^2 denote the first and second derivatives respectively.

II. DEFINITIONS AND BACKGROUND

Definition 1. *A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be α- gradient Lipschitz if for any $\boldsymbol{x_1}, \boldsymbol{x_2}$ in the domain of f, and for $\alpha > 0$,*

$$\|\nabla f(\boldsymbol{x_1}) - \nabla f(\boldsymbol{x_2})\| \leq \alpha \|\boldsymbol{x_1} - \boldsymbol{x_2}\|, \tag{1}$$

where α is known as the gradient Lipschitz constant. The smallest such constant is known as the optimal constant, denoted by α^*.

Nesterov's seminal work [27] discusses the following theorem which guarantees the convergence of the GD algorithm.

Lemma 1. *[27] For an α-gradient Lipschitz function $f : \mathbb{R}^d \to \mathbb{R}$, gradient descent with a step size $\eta \leq 1/\alpha$ produces a decreasing sequence of objective values and the optimal step size is given by $\eta^* = 1/\alpha$.*

For a doubly differentiable function f with gradient Lipschitz constant as α, we have [27]

$$\nabla^2 f(\boldsymbol{x}) \preceq \alpha \boldsymbol{I} \quad \forall x. \tag{2}$$

This implies that all eigenvalues of the matrix $\nabla^2 f(\boldsymbol{x}) - \alpha \boldsymbol{I}$ should be less than or equal to zero for all values of $\boldsymbol{x}$. This is achieved when the maximum eigenvalue satisfies this condition. Therefore, the gradient Lipschitz constant of a double differentiable function is given by

$$\alpha^* = \max_{\boldsymbol{x}} \lambda_{max}(\nabla^2 f(\boldsymbol{x})). \tag{3}$$

We use (3) in the following sections to derive the required constant. Note that any $\alpha > \alpha^*$ also satisfies (2). Therefore, if the exact value for α^* cannot be determined, an upper bound on α^* can be derived. The learning rate derived from the upper bound also results in a decreasing sequence of iterates according to Lemma 1. This signifies that the learning rate derived as the inverse of the gradient Lipschitz constant or any upper bound will always result in convergence of the gradient descent algorithm. This implication is used by us to guarantee the convergence of GD while training neural networks.

III. DERIVING THE GRADIENT LIPSCHITZ CONSTANT FOR A SINGLE HIDDEN LAYER NEURAL NETWORK

In this section, a neural network with a single hidden layer consisting of k neurons with activation function $act(.)$ is considered, as given in Fig. 1. We derive the gradient Lipschitz constant for two different popular activation functions: sigmoid and ReLU. The weight vector from the input to the jth hidden layer neuron is denoted as $\boldsymbol{w}^j$ where $\boldsymbol{w}^j \in \mathbb{R}^d$ for $j = 1, ...k$. The column vector $\boldsymbol{w}$ refers to the stack of vectors $\boldsymbol{w}^1, ...\boldsymbol{w}^k$; $\boldsymbol{w} \in \mathbb{R}^{kd}$. The output of the network is taken as the sum of outputs from each of the hidden layer neurons and is given by $f(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^k act(\boldsymbol{x}^T \boldsymbol{w}^j)$ for input $\boldsymbol{x}$. The training data is denoted as a set of points $(\boldsymbol{x}(i), y(i))$ for $i = 1, ...N$. The aim of the network is to learn the function f given the training data. Throughout, we consider the quadratic loss function namely,

$$l(\boldsymbol{w}) = \frac{1}{2N} \sum_{i=1}^N \left(\left(\sum_{j=1}^k act(\boldsymbol{x}(i)^T \boldsymbol{w}^j) \right) - y(i) \right)^2. \tag{4}$$
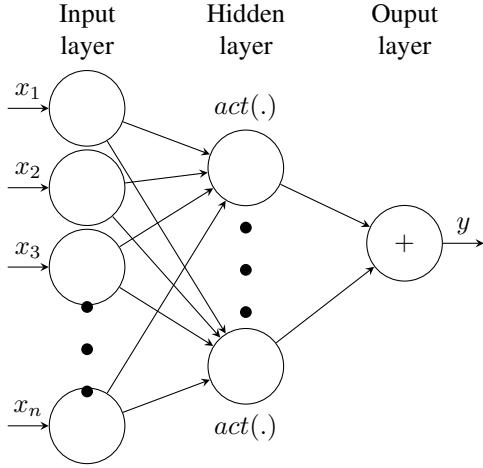
Fig. 1: The architecture of a single hidden layer network

A. Sigmoid activation

The sigmoid activation is defined as $\sigma(x) = \frac{1}{1+\exp(-x)}$. The gradient Lipschitz constant for a single hidden layer network with sigmoid activation function is derived in this section. Initially, we consider a single data point, $(\boldsymbol{x}, y)$, and then extend it to a database.

Theorem 1. *The gradient Lipschitz constant for a single-hidden layer feedforward network with sigmoid activation when considering quadratic loss function in (4) with $act(.) = \sigma(.)$ and $N = 1$ is given by,*

$$\alpha^* \leq \min\left(\frac{|k - y|}{10} + \frac{k}{16}, 0.1176(k-1) + \frac{|y|}{10} + 0.077\right)\|\boldsymbol{x}\|^2. \tag{5}$$

Proof: As the loss function is doubly differentiable, the required constant is $\alpha^* = \max_{\boldsymbol{w}} \lambda_{max}(\nabla^2 l(\boldsymbol{w}))$. Note,

$$\nabla l(\boldsymbol{w}) = \left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}^T\boldsymbol{w}^j) - y\right)\begin{bmatrix}\sigma(\boldsymbol{x}^T\boldsymbol{w^1})(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w^1}))\boldsymbol{x} \\ \vdots \\ \sigma(\boldsymbol{x}^T\boldsymbol{w^k})(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w^k}))\boldsymbol{x}\end{bmatrix}. \tag{6}$$

$$\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w}) \triangleq \begin{bmatrix}\sigma(\boldsymbol{x}^T\boldsymbol{w^1})(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w^1})) \\ \vdots \\ \sigma(\boldsymbol{x}^T\boldsymbol{w^k})(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w^k}))\end{bmatrix}, \tag{7}$$

is defined where $\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w}) \in \mathbb{R}^k$. Let $Diag_m(k_m)$ denote a diagonal matrix whose non-zero entry in the m_{th} row is k_m. The Hessian matrix computed using the product rule of differentiation is given by,

$$\nabla^2 l(\boldsymbol{w}) = \left(Diag_m\left(\left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}^T\boldsymbol{w}^j) - y\right)\sigma(\boldsymbol{x}^T\boldsymbol{w}^m)\right.\right.$$
$$\left.(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w}^m))(1 - 2\sigma(\boldsymbol{x}^T\boldsymbol{w}^m))\right)$$
$$\left.+ \hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})^T\right) \otimes \boldsymbol{x}\boldsymbol{x^T}. \tag{8}$$

The gradient Lipschitz constant is given by

$$\alpha^* = \max_{\boldsymbol{w}} \lambda_{max}\left[\left(Diag_m\left(\left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}^T\boldsymbol{w}^j) - y\right)\sigma(\boldsymbol{x}^T\boldsymbol{w}^m)\right.\right.\right.$$
$$\left.(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w}^m))(1 - 2\sigma(\boldsymbol{x}^T\boldsymbol{w}^m))\right)$$
$$\left.\left.+ \hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})^T\right) \otimes \boldsymbol{x}\boldsymbol{x^T}\right]. \tag{9}$$

Note (9) involves a maximization over all possible values of $\boldsymbol{w}$ and an eigenvalue computation for every value. We use the structure of the matrix to provide a simplified solution. We use the following property of Kronecker products [30].

Lemma 2. *Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ have eigenvalues $\lambda_i, i \in n$, and let $\boldsymbol{B} \in \mathbb{R}^{m \times m}$ have eigenvalues $\mu_j, j \in m$, then the mn eigenvalues of $\boldsymbol{A} \otimes \boldsymbol{B}$ are given by*

$$\lambda_1\mu_1, \cdots, \lambda_1\mu_m, \lambda_2\mu_1, \cdots, \lambda_2\mu_m, \cdots, \lambda_n\mu_m.$$

Therefore, the maximum eigenvalue of the Kronecker product will be the product of the maximum eigenvalues, if the maximum eigenvalue of the diagonal matrix is positive; else, it will be zero. As we are maximizing over all possible values of $\boldsymbol{w}$, we can always ensure that the maximum eigenvalue is positive. Since $\boldsymbol{x}\boldsymbol{x^T}$ is rank one with a single non-zero eigenvalue, $\boldsymbol{x^T}\boldsymbol{x}$, using Lemma 2, we have,

$$\alpha^* = \max_{\boldsymbol{w}} \lambda_{max}(\boldsymbol{P})\boldsymbol{x^T}\boldsymbol{x}, \tag{10}$$

where $\boldsymbol{P}$ is defined as

$$\boldsymbol{P} \triangleq Diag_m\left(\left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}^T\boldsymbol{w}^j) - y\right)\sigma(\boldsymbol{x}^T\boldsymbol{w}^m)\right.$$
$$\left.(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w}^m))(1 - 2\sigma(\boldsymbol{x}^T\boldsymbol{w}^m))\right)$$
$$+ \hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})^T. \tag{11}$$

A bound can be obtained to find the maximum eigenvalue of $\boldsymbol{P}$ using the Weyl's inequality which states that for Hermitian matrices $\boldsymbol{A}$ and $\boldsymbol{B}$,

$$\lambda_{max}(\boldsymbol{A} + \boldsymbol{B}) \leq \lambda_{max}(\boldsymbol{A}) + \lambda_{max}(\boldsymbol{B}). \tag{12}$$

Using the above inequality, the observation that $\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})^T$ is rank-1 and that the eigenvalues of a diagonal matrix are the diagonal entries, one obtains,

$$\lambda_{max}(\boldsymbol{P}) \leq \max_{m}\left(\left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}^T\boldsymbol{w}^j) - y\right)\sigma(\boldsymbol{x}^T\boldsymbol{w}^m)\right.$$
$$\left.(1 - \sigma(\boldsymbol{x}^T\boldsymbol{w}^m))(1 - 2\sigma(\boldsymbol{x}^T\boldsymbol{w}^m))\right)$$
$$+ \hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w})^T\hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w}). \tag{13}$$

Combining (10) and (13),

$$\alpha^* \leq \max_{\boldsymbol{w}} \Bigg(\max_m \Bigg(\Big(\sum_{j=1}^k \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) - y \Big)$$

$$\sigma(\boldsymbol{x}^T \boldsymbol{w}^m)(1 - \sigma(\boldsymbol{x}^T \boldsymbol{w}^m))(1 - 2\sigma(\boldsymbol{x}^T \boldsymbol{w}^m)) \Bigg)$$

$$+ \left\| \hat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{w}) \right\|^2 \Bigg) \|\boldsymbol{x}\|^2. \tag{14}$$

The expression in (14) can be written in terms of the derivatives of sigmoid function as given below:

$$\alpha^* \leq \max_{\boldsymbol{w}} \Bigg(\max_m \Bigg(\Big(\sum_{j=1}^k \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) - y \Big)$$

$$\nabla^2 \sigma(\boldsymbol{x}^T \boldsymbol{w}^m) \Bigg) + \sum_{j=1}^k \big(\nabla \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) \big)^2 \Bigg) \|\boldsymbol{x}\|^2. \tag{15}$$

We now use the following bounds on the sigmoid derivatives [31] to bound (15):

$$0 \leq \sigma(x) \leq 1 \quad \forall x \tag{16}$$

$$\nabla_x \sigma(x) = \sigma(x)(1 - \sigma(x)) \leq \frac{1}{4} \quad \forall x \tag{17}$$

$$\nabla_x^2 \sigma(x) = \sigma(x)(1 - \sigma(x))(1 - 2\sigma(x)) \leq \frac{1}{10} \quad \forall x. \tag{18}$$

Using the above conditions to individually maximize each of the terms in (15),

$$\alpha^* \leq \left[\frac{|k - y|}{10} + \frac{k}{16} \right] \|\boldsymbol{x}\|^2. \tag{19}$$

We note that tighter bounds may be achieved by maximizing (15) as a whole instead of each individual term. As the maximization in (15) is over the weights $\boldsymbol{w}$, considering the terms consisting of $\boldsymbol{w}$,

$$\max_m \Bigg(\Big(\sum_{j=1}^k \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) - y \Big) \nabla^2 \sigma(\boldsymbol{x}^T \boldsymbol{w}^m) \Bigg) +$$

$$\sum_{j=1}^k \big(\nabla \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) \big)^2 \tag{20}$$

Note that maximizing (20) with respect to $\boldsymbol{w}$ maximizes (15). Let us assume that the index that maximizes the inner maximization with respect to m is $\bar{m}$. Therefore, (20) is now rewritten as,

$$\Big(\sum_{j=1}^k \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) - y \Big) \nabla^2 \sigma(\boldsymbol{x}^T \boldsymbol{w}^{\bar{m}}) + \sum_{j=1}^k \big(\nabla \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) \big)^2. \tag{21}$$

We use $a - b \leq |a| + |b|$ on the first term. Combining the terms corresponding to $\bar{m}$ and using (18) to bound the second derivative,

$$\max_{\boldsymbol{w}} \Bigg[\frac{1}{10} \Big(\sum_{j=1, j \neq \bar{m}}^k \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) + |y| \Big) + \sum_{j=1, j \neq \bar{m}}^k \big(\nabla \sigma(\boldsymbol{x}^T \boldsymbol{w}^j) \big)^2$$

$$+ \sigma(\boldsymbol{x}^T \boldsymbol{w}^{\bar{m}}) \nabla^2 \sigma(\boldsymbol{x}^T \boldsymbol{w}^{\bar{m}}) + \big(\nabla \sigma(\boldsymbol{x}^T \boldsymbol{w}^{\bar{m}}) \big)^2 \Bigg]. \tag{22}$$

We then maximize each of these terms individually leading to the following bounds:

- $\frac{\sigma(x)}{10} + (\nabla \sigma(x))^2 \leq 0.1176 \quad \forall x$
- $\sigma(x) \nabla^2 \sigma(x) + (\nabla \sigma(x))^2 \leq 0.0770 \quad \forall x.$

Incorporating the above, we get the following bound on the gradient Lipschitz constant

$$\alpha^* \leq \left[(k-1)0.1176 + \frac{|y|}{10} + 0.0770 \right] \|\boldsymbol{x}\|^2. \tag{23}$$

Depending on the value of k and y, we find that either of the bounds in (19) and (23) can prove tighter. As both of them are upper bounds, we pick the least one of them. The final expression for the upper bound on the gradient Lipschitz constant when a single data point $(\boldsymbol{x}, y)$ is taken is given by,

$$\alpha^* \leq \min\Big(\frac{|k - y|}{10} + \frac{k}{16}, 0.1776(k-1)$$

$$+ \frac{|y|}{10} + 0.0770 \Big) \|\boldsymbol{x}\|^2. \tag{24}$$

$\blacksquare$

We now wish to extend this to multiple data points $(\boldsymbol{x}(i), y(i))$ for $i = 1, \cdots N$. When we follow the same derivation for a loss function constructed with multiple data points, the derived upper bound results in the average of the individual upper bounds. This is a direct implication from the fact that $\nabla^2 \big(\sum_i f_i \big) = \sum_i \nabla^2 f_i$.

Therefore, the bound on α^* is given by

$$\alpha^* \leq \frac{1}{N} \sum_{i=1}^N \min \Bigg[\frac{|k - y(i)|}{10} + \frac{k}{16},$$

$$0.1776(k-1) + \frac{|y(i)|}{10} + 0.0770 \Bigg] \|\boldsymbol{x}(i)\|^2. \tag{25}$$

As the loss function for multiple data points is defined as the average over loss using each of the data points, we note that the derived upper bound on the gradient Lipschitz constant also follows a similar structure. Given the number of neurons in the hidden layer, k, and the data set, the upper bound can be found by simply evaluating the expression derived in (25); the inverse of this bound gives a learning rate which always guarantees that GD will converge. The bound increases with increase in width of the network as well as the norm of the input. It is noted that the bound depends on data only through its norm. Therefore, if different data sets with similar Euclidean norms are encountered, the derived bound can simply be reused.

B. ReLU activation

The ReLU activation function is given by $s(x) = \max(0, x)$. Initially, consider a single data point $(\boldsymbol{x}, y)$ for deriving the gradient Lipschitz constant.

Theorem 2. *The gradient Lipschitz constant for a single-hidden layer feedforward network with ReLU activation when considering quadratic loss function in (4) when $act(.) = s(.)$ and $N = 1$ is given by*

$$\alpha^* = k \|\boldsymbol{x}\|^2. \tag{26}$$

Proof: Please see Appendix A.

$\blacksquare$

We now extend the derivation of the gradient Lipschitz constant for a multiple input database. The result in Theorem 2 can be extended to N inputs as

$$\alpha^* = \frac{1}{N} \max_{\boldsymbol{w}} \lambda_{\max} \left(\sum_{i=1}^{N} \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})^T \right), \quad (27)$$

where

$$\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) \triangleq \left[\mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^1 \geq 0\}} \boldsymbol{x} \quad \dots \quad \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^k \geq 0\}} \boldsymbol{x} \right]^T. \quad (28)$$

This involves a maximization over all possible weights and we would like to derive a closed form expression. It is observed that the Hessian matrix in this specific problem is structured; it is the sum of outer products of the vector $\boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})$ where the vector consists of $\boldsymbol{x}(i)$ multiplied by appropriate indicators. We wish to exploit the structure of the Hessian matrix to arrive at an elegant solution which can be easily evaluated. Towards that end, we state and prove the following lemma.

Lemma 3. *For a vector $\boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})$ as defined in (28), the following relation holds*

$$\lambda_{max} \left(\sum_{i=1}^{N} \bar{\boldsymbol{a}}(\boldsymbol{x}(i)) \bar{\boldsymbol{a}}(\boldsymbol{x}(i))^T \right) \geq$$

$$\lambda_{max} \left(\sum_{i=1}^{N} \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})^T \right) \quad \forall \boldsymbol{w} \quad (29)$$

where

$$\bar{\boldsymbol{a}}(\boldsymbol{x}) \triangleq \begin{bmatrix} \boldsymbol{x} & \dots & \boldsymbol{x} \end{bmatrix}^T \quad \text{(k terms)}. \quad (30)$$

Proof: Please see Appendix B. ∎

Lemma 3 holds for all values of $\boldsymbol{w}$; therefore, it also holds for that $\boldsymbol{w}$ which maximizes the maximum eigenvalue in (27). In essence, Lemma 3 provides an upper bound on the constant α^*. It is also noted that as $\bar{\boldsymbol{a}}(\boldsymbol{x}(i))$ is an instance of $\boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})$ for a specific $\boldsymbol{w}$,

$$\max_{\boldsymbol{w}} \lambda_{max} \left(\sum_{i=1}^{N} \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}(i), \boldsymbol{w})^T \right) \geq$$

$$\lambda_{max} \left(\sum_{i=1}^{N} \bar{\boldsymbol{a}}(\boldsymbol{x}(i)) \bar{\boldsymbol{a}}(\boldsymbol{x}(i))^T \right). \quad (31)$$

Hence, (31) gives a lower bound on the constant α^*. From (29) and (31), it is evident that the upper and the lower bounds coincide and must be equal to the exact value of α^*, i.e.,

$$\alpha^* = \lambda_{max} \left(\frac{1}{N} \sum_{i=1}^{N} \bar{\boldsymbol{a}}(\boldsymbol{x}(i)) \bar{\boldsymbol{a}}(\boldsymbol{x}(i))^T \right). \quad (32)$$

The exact gradient Lipschitz constant for a single hidden layer network with ReLU activation has been derived in (32). We no longer need the perform the brute force maximization over all weight values as was required in (27). Instead evaluating α^* is now reduced to finding the maximum eigenvalue of a $kd \times kd$ matrix. We note that the value of α^* only depends on the data vectors $\boldsymbol{x}(i)$ and the number of neurons k. As the dimension of the problem increases, the eigenvalue computation will get intensive; in such cases, we can employ well-established bounds like Gershgorin and Brauer's ovals of
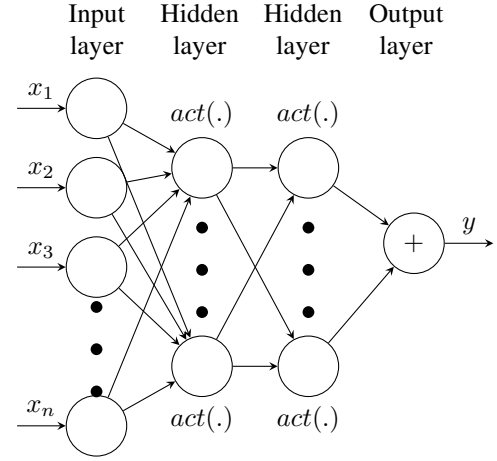
Cassini to provide easily computable upper bounds on α^*. For convenience, these theorems are stated here.

Theorem 3 (Gershgorin's Circles theorem [32])**.** *For a square matrix $\boldsymbol{A}$, the upper bound on the maximum eigenvalue is,*

$$\lambda_{max}(A) \leq \max_i (a_{ii} + R_i(\boldsymbol{A})), \quad (33)$$

where $R_i(\boldsymbol{A}) = \sum_{i \neq j} |a_{ij}|$

Theorem 4 (Brauer's Ovals of Cassini)**.** *For a square matrix $\boldsymbol{A}$, the upper bound on the maximum eigenvalue is given by*

$$\lambda_{max}(A) \leq \max_{i \neq j} \left(\frac{a_{ii} + a_{jj}}{2} + \sqrt{(a_{ii} - a_{jj})^2 + R_i(\boldsymbol{A}) R_j(\boldsymbol{A})} \right), \quad (34)$$

where $R_i(\boldsymbol{A}) = \sum_{i \neq j} |a_{ij}|$.

The bound in Theorem 4 is guaranteed to be provide a bound which is not worse than the Gershgorin bound [33]. The bounds stated above can be used to provide an upper bound on the gradient Lipschitz constant if eigenvalue computation is a constraint. The inverse of the derived constant α^* or its upper bound can be used as the learning rate while training the network, and this will guarantee convergence of GD.

IV. DERIVING THE GRADIENT LIPSCHITZ CONSTANT FOR A TWO HIDDEN LAYER NEURAL NETWORK

Here, we focus on a shallow architecture with two hidden layers between the input and output layers as illustrated in Fig. 2. The weight matrix between the input and the first hidden layer is denoted as $\boldsymbol{V} \in \mathbb{R}^{d \times k_1}$ where k_1 is the number of neurons in the first hidden layer. The weight matrix between the two hidden layers is denoted as $\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2}$ where k_2 is the number of neurons in the second hidden layer. The output of the network is the sum of the outputs of the neurons in the second hidden layer. Let us denote the parameters of the network $\boldsymbol{V}, \boldsymbol{W}$ as a single vector $\boldsymbol{\theta}$. Note that the dimension of $\boldsymbol{\theta}$ is $k_1(d + k_2)$.

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{V^1}^T \dots \boldsymbol{V^{k_1}}^T & \boldsymbol{W^1}^T \dots \boldsymbol{W^{k_2}}^T \end{bmatrix}^T \quad (35)$$

Fig. 2: The architecture of a two hidden layer network

This derivation is challenging as it is not a straight-forward extension of the single layer case; the Hessian involves two weight matrices $\boldsymbol{V}, \boldsymbol{W}$ to be optimized over. The squared loss function is considered whose expression is given by,

$$l(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left(\left(\sum_{l_2=1}^{k_2} act \left(\sum_{l_1=1}^{k_1} act(\boldsymbol{x}(i)^T \boldsymbol{V}^{\boldsymbol{l_1}}) \right. \right. \right.$$
$$\left. \left. \left. W_{l_1 l_2} \right) \right) - y(i) \right)^2. \quad (36)$$

A. Sigmoid activation

Here, we derive the gradient Lipschitz constant of a 2-hidden layer network with sigmoid activation function. As done previously, we initially consider a single data tuple $(\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$.

Theorem 5. *The gradient Lipschitz constant for a two hidden layer feedforward network with sigmoid activation when considering quadratic loss function in (36) with $act(.) = \sigma(.)$ and $N = 1$ is given by,*

$$\alpha^* \leq k_1 \left(\frac{k_2 \beta \|\boldsymbol{x}\|}{16} \right)^2 + \frac{k_1 k_2}{16} + \max \left(\frac{1}{10} + \right.$$
$$\left[\frac{1}{4} + \frac{\beta}{10} \right] \frac{k_1 \|\boldsymbol{x}\|_1}{4}, \left[\frac{1}{4} + \frac{\beta}{10} \right] \frac{k_2 \|\boldsymbol{x}\|_\infty}{4} +$$
$$\left[\frac{\beta}{1000} + \frac{1}{4} \right] k_1 k_2 \beta \|\boldsymbol{x}\|_1 \|\boldsymbol{x}\|_\infty \right) |k_2 - y| \quad (37)$$

when $|\theta_i| < \beta \quad \forall i$ for $\boldsymbol{\theta}$ as defined in (35).

Proof: Please see Appendix C. ∎

This is further extended to the case of N inputs and the obtained constant is given by

$$\alpha^* \leq \frac{1}{N} \sum_{i=1}^{N} \left[k_1 \left(\frac{k_2 \beta \|\boldsymbol{x}(i)\|}{16} \right)^2 + \frac{k_1 k_2}{16} + \max \left(\frac{1}{10} + \right. \right.$$
$$\left[\frac{1}{4} + \frac{\beta}{10} \right] \frac{k_1 \|\boldsymbol{x}(i)\|_1}{4}, \left[\frac{1}{4} + \frac{\beta}{10} \right] \frac{k_2 \|\boldsymbol{x}(i)\|_\infty}{4} +$$
$$\left. \left[\frac{\beta}{1000} + \frac{1}{4} \right] k_1 k_2 \beta \|\boldsymbol{x}(i)\|_1 \|\boldsymbol{x}(i)\|_\infty \right) |k_2 - y(i)| \right].$$
$$(38)$$

We note that increase in dimension of the architecture will lead to an increase in the bound. The derived bound also depends on the maximum value in the weight matrix. Therefore, the bound is tighter when there are no spurious values with large magnitude in the weight matrix.

B. ReLU Activation

Initially, consider a single data tuple $(\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$.

Theorem 6. *The gradient Lipschitz constant for a two hidden layer feedforward network with ReLU activation when considering quadratic loss function in (36) with $act(.) = s(.)$ and $N = 1$ is given by*

$$\alpha^* \leq k_1(d + k_2)\beta^2 \|\boldsymbol{x}\|^2 + \max(A_{max} k_2 |\boldsymbol{x}|_\infty, A_{max} |\boldsymbol{x}|_1),$$
$$(39)$$

where $A_{max} = k_1 k_2 \beta^2 \|\boldsymbol{x}\| - y$ when $|\theta_i| < \beta \quad \forall i$.

Proof: Please see Appendix D. ∎

Extending to a database of N inputs, i.e., $(\boldsymbol{x}(i), y(i))$ for $i = 1, ...N$, the following bound is obtained on the gradient Lipschitz constant,

$$\alpha^* \leq \frac{1}{N} \sum_{i=1}^{N} \left(k_1(d + k_2)\beta^2 \|\boldsymbol{x}(i)\|^2 + \right.$$
$$\left. \max((A_{max}(i) k_2 |\boldsymbol{x}(i)|_\infty, A_{max}(i) |\boldsymbol{x}(i)|_1) \right), \quad (40)$$

where $A_{max}(i) = k_1 k_2 \beta^2 \|\boldsymbol{x}(i)\| - y(i)$. The derived bound depends on the dimension of the problem and on the factor β which is the maximum magnitude in the weight matrix. The bound increases linearly with increase in any of the following parameters: k_1, k_2, d and quadratically on β.

V. PROPOSED SEARCH ALGORITHM

We propose an algorithm that uses the derived bounds from previous sections to arrive at a learning rate which exhibits faster convergence than using the inverse of the derived bound.

A. Why is a search algorithm required?

For a derived upper bound α, the corresponding learning rate is found as $\eta = 1/\alpha$. Note that $\eta < \eta^*$ (where $\eta^* = 1/\alpha^*$) and therefore, any learning rate derived from an upper bound is guaranteed to result in non-increasing traces for GD. However, there may exist learning rates that are greater than η which lead to faster convergence.

Even when the exact value of gradient Lipschitz constant is available, optimality over all possible initializations is considered. However, in practical scenarios, the range of values with which the neural networks are initialized are restricted and hence, we do not require a universally optimal learning rate. In other words, we can afford to have learning rates even higher than η^* as long as it guarantees monotonically decreasing iterates in the region where the weights are initialized.

Summarizing, the motivations for proposing a search are two-fold: the derived bounds may be loose which gives room for finding better learning rates, and we wish to exploit the weight initialization to find a learning rate customized to the initialization.

B. Proposed algorithm

The search is for a learning rate which leads to faster convergence than the inverse of the derived bound, while ensuring that it produces decreasing iterates. This search can be conducted by employing a search interval customized for a given data set and weight initialization. The start-of-the-art hyper-parameter tuning libraries such as HyperOpt [14] allow the user to set the search space. In our work, we adopt the HyperOpt[1] implementation of TPE [14]. As they do not utilize the information regarding the objective, the search for learning rate is typically conducted in the interval $[0, 1]$.

The algorithm is inspired from binary search [34]. The

---

[1]In this manuscript, we refer to the TPE implementation in the HyperOpt library as simply HyperOpt.

Algorithm 1 Binary search algorithm

1: **Input**: Derived bound α, Evaluations E, Epochs T
2: **Initialization:** $\eta = 1/\alpha$, $l_c = \alpha$, $loss^* = Loss(\eta, T)$
3: **for** $i = 1, 2, \cdots E$ **do**
4:      Run GD for and observe $Loss(\eta, T)$.
5:      **if** $Loss(\eta) < loss^*$ AND Iterates are non-increasing **then**
6:          $loss^* = Loss(\eta, T)$ (Update best loss)
7:          $l_c = \eta^{-1}$
8:          $\eta = (l_c/2)^{-1}$ (Increase learning rate)
9:      **else**
10:         $\eta = \left[\dfrac{\eta^{-1} + l_c}{2} \right]^{-1}$ (Decrease learning rate)
11:      **end if**
12: **end for**
13: **Output:** Learning rate = η

---

algorithm is initialized with a learning rate that is guaranteed to converge (i.e., $1/\alpha$ where α is the derived bound) and is allowed a certain number of trials. If the learning rate chosen in a trial results in a converging trace of GD, a higher learning rate is chosen for the next trial; else, a lower learning rate is chosen. The learning rate leading to the lowest loss is reported at the end of the search algorithm. We note that as the algorithm is initialized with a convergent learning rate, it never yields a divergent learning rate, unlike other search algorithms like grid search, random search and HyperOpt.

Note that one can apply more sophisticated search techniques to carry out this search; we adopt the BinarySearch algorithm as it is intuitive and effective. When the ends of a search interval are known, binary search is typically employed in many applications such as [35]. In our implementation, the BinarySearch algorithm checks if the midpoint of the interval results in a learning rate that gives monotonically decreasing iterates. If so, it searches through the lower interval, else, it chooses the higher interval.

The proposed algorithm is described in Algorithm 1. Note that in the algorithm, $Loss(\eta, T)$ refers to the value of the loss function at the end of T epochs using the learning rate η.

C. Advantages and remarks

The inverse of the derived gradient Lipschitz constant always acts as a valid learning rate. Therefore, in applications where a slower convergence is acceptable, this method is highly useful since it allows one to actually skip hyperparameter tuning altogether.

In other search methods, the search space is often considered as $[0, 1]$. However, there may be applications where the inverse of the gradient Lipschitz constant is greater than one. This in turn implies that the proposed method will choose learning rates greater than one whilst guaranteeing convergence whereas the traditional methods with restricted search space, say $[0, 1]$ will choose a learning rate less than 1.

In the case that the optimal learning rate is of a very low order, search algorithms like random search or HyperOpt may always encounter diverging behaviour even after the allotted number of evaluations are utilized. However, in the case of

the proposed BinarySearch algorithm, we are guaranteed to find a learning rate which would result in a successful GD epoch. These advantages are demonstrated with the help of simulations in the forthcoming section.

VI. SIMULATION RESULTS

The effectiveness of the proposed algorithm is compared against HyperOpt. As HyperOpt is already shown to outperform random search [14], we only compare with the HyperOpt tool that uses the TPE. To do so, we run 100 experiments with the same number of evaluations allotted for both HyperOpt and BinarySearch. To compare optimization strategies, we can opt for any of the following metrics[2]:

- Best-found value: The loss achieved during the best-performing evaluation in an experiment is compared and the fraction of times BinarySearch outperforms HyperOpt is tabulated.
- Best trace: The best trace for both the competing algorithms are compared. The learning rate leading to the least area under the convergence curve is said to yield the best trace.

Note that using best loss as the only metric for comparing two optimization techniques may not be sufficient. For example, consider two optimization mechanisms that reach the same minimum in 100 and 1000 epochs. The best loss metric ranks both algorithms equally whereas convergence in 100 epochs is preferable. As the best trace metric compares the area below the convergence curves, it ranks the algorithm using 100 epochs higher than the other. The speed of convergence especially becomes important while training complex models which take significant amount of time to train. The synthetic simulation is inspired from the setting in works like [19], [20] that deal with the theoretical properties of shallow networks. We consider a database with points $(\boldsymbol{x}(i), y(i))$ for $i = 1, ..N$ where $\boldsymbol{x}(i) \sim \mathcal{N}(\boldsymbol{0}, I)$ similar to [19]. It is assumed that there is an underlying network known as the teacher network with weights $\boldsymbol{w}^*$. The weights of the teacher network are also sampled from a zero mean unit variance Gaussian distribution. The corresponding labels $y(i)$ are generated by passing the data through the teacher network. For our simulations, we consider $N = 100$ with T epochs.

The network to be trained is referred to as the student network. The weights of the student network are initialized using Xavier initialization [36] and the quadratic loss function is employed. The optimization algorithm used for training is GD and it is run for T epochs. The algorithms, both BinarySearch and HyperOpt, are allowed a fixed number of evaluations. This is repeated for 100 experiments (each with a different database and weight initialization). All results reported are over 100 experiments.

A. One hidden layer networks

1) Comparison with HyperOpt: For a single hidden layer, we run GD for $T = 100$ epochs. We note that the best learning rate chosen by HyperOpt after the stipulated number of evaluations sometimes still lead to unsuccessful GD epochs

---

[2]https://sigopt.com/blog/evaluating-hyperparameter-optimization-strategies/

in case of ReLU activation, i.e., the iterates diverge while our method never leads to divergent behaviour. The fraction of times that divergent behaviour is observed for HyperOpt is tabulated in Table I. In the remaining successful experiments, we compare the final loss obtained using the learning rate chosen by both BinarySearch and HyperOpt. The fraction of experiments in which BinarySearch outperforms (results in a lower 'best-found value' than) HyperOpt is tabulated in Table II.

		No. of evaluations		
d	k	5	10	20
10	10	0.12	0.01	0
20	5	0.03	0	0
5	20	0.18	0.07	0
20	20	0.37	0.07	0.02

TABLE I: Fraction of times HyperOpt diverges for 1 hidden layer network with ReLU activation

		ReLU activation			Sigmoid activation		
		No. of evaluations			No. of evaluations		
d	k	5	10	20	5	10	20
10	10	0.81	0.91	0.93	1	1	1
20	5	0.76	0.85	0.85	1	1	1
5	20	0.74	0.90	0.95	1	1	1
20	20	0.73	0.86	0.89	1	1	1

TABLE II: Fraction of times the best value for BinarySearch outperforms HyperOpt for 1 hidden layer network out of successful experiments[3]

We notice that for higher number of evaluations, BinarySearch always outperforms HyperOpt. It should be noted that these comparisons are performed after eliminating the experiments for which HyperOpt diverges. For instance, for the configuration $d = k = 20$ with 5 evaluations using ReLU activation, BinarySearch outperforms HyperOpt 73% out of the $100 - 37 = 63$ successful experiments. If we also consider the divergent experiments, BinarySearch outperforms HyperOpt 83% of the times. In the case of sigmoid activation, the divergent behaviour is not observed. As the gradient of the loss function is of a small order of magnitude (in the order of 10^{-2}), GD does not diverge for higher learning rates. Also, the learning rate derived as the inverse of gradient Lipschitz constant for $d = 2, k = 3, N = 100$ is 1.78 which itself is greater than 1. This implies that any learning rate less than 1.78 will never lead to divergent behaviour and learning rates greater than 1.78 can be explored. One can argue that the derived bound can be used to modify the search interval of existing algorithms; this is discussed at the end of this section.

The metric, 'best-found value' grades an algorithm based on the final loss value that the algorithm converges to. We also need a metric to quantify the performance in terms of the convergence rate. Hence, we also provide the best-trace metric, where the tuning strategies are compared based on their convergence. The curve with the fastest convergence (least area under convergence curve) out of all 100 experiments is plotted for BinarySearch and HyperOpt. We provide the

[3]Experiments in which HyperOpt diverges are not considered.

results for a specific configuration with $d = k = 10$ where each method is allowed 10 evaluations for ReLU and sigmoid activation functions in Fig. 3 and 4 respectively. We note that the proposed method results in better convergence curves than the existing method, HyperOpt.
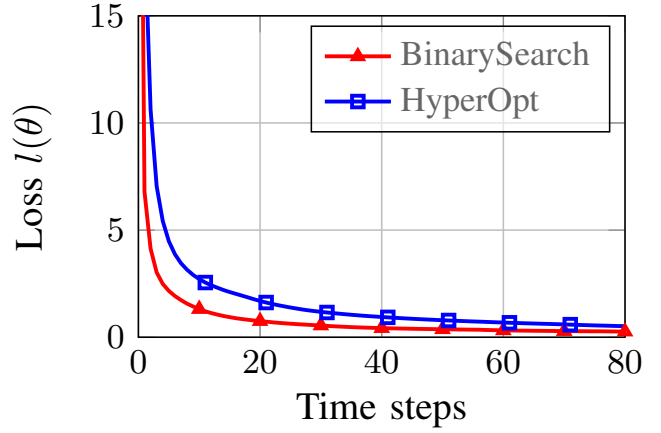

Fig. 3: Best trace comparison of single hidden layer ReLU network with $d = k = 10$ with 10 evaluations
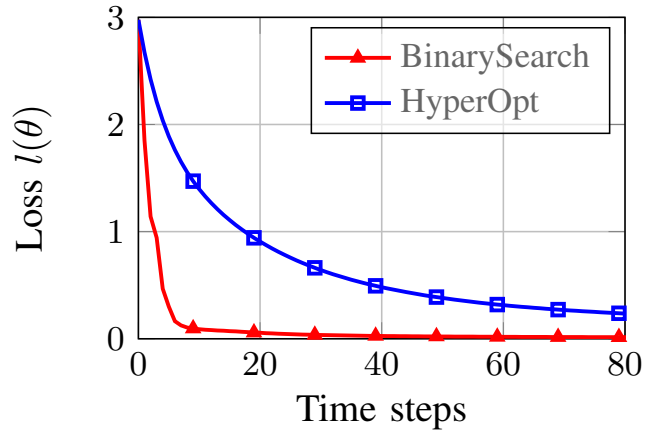

Fig. 4: Best trace comparison of single hidden layer sigmoid network with $d = k = 10$ with 10 evaluations

In order to further study the attributes of the learning rates chosen by the proposed method in comparison with HyperOpt, we tabulate the mean, standard deviation, maximum and minimum values chosen over the 100 experiments. We study this for the case of $d = k = 10$ in Table III.

Activation	Evals	Algorithm	Mean	Std. dev	Max	Min
ReLU	5	BinarySearch	0.324	0.033	0.409	0.227
		HyperOpt	0.234	0.095	0.459	0.017
	20	BinarySearch	0.385	0.045	0.498	0.292
		HyperOpt	0.328	0.069	0.462	0.098
Sigmoid	5	BinarySearch	3.175	0.071	3.327	2.983
		HyperOpt	0.778	0.174	0.996	0.342
	20	BinarySearch	13.359	3.984	30.805	7.514
		HyperOpt	0.868	0.095	0.996	0.565

TABLE III: Variation of the chosen learning rates for single hidden layer network

Although the above tabulation is for 100 experiments, note that HyperOpt returns a learning rate that results in diverging traces for a small fraction of experiments (0.01); these entries are ignored while computing the tabulated constants for HyperOpt.

From Table III, it is noted that BinarySearch always chooses a larger learning rate on an average as compared to HyperOpt which leads to better convergence. The maximum learning rate (the learning rate yielding the best trace graph) chosen by BinarySearch is greater than that of HyperOpt as the number of evaluations increase as evidenced by the numbers corresponding to ReLU activation. In the case of ReLU activation, it is observed that the proposed method has lesser variance in choosing a step size as compared to HyperOpt. For the sigmoid activation, our method chooses rates much greater than one, as it is allowed by the structure of the problem whereas HyperOpt typically is restricted to the interval $[0, 1]$. This also explains why BinarySearch results in a much faster convergence than HyperOpt in this case.

2) Comparison with other optimization algorithms: We also compare the performance of our proposed tuning method against popular optimization algorithms such as Adam [7], Adagrad [37], Adadelta [38] and RMSProp [39]. Although a default learning rate of 0.001 is suggested for Adam, RMSProp and Adadelta, we note that a learning rate of 0.01 fares better in this case. To demonstrate, we compare against the default learning rate (0.001) as well as a learning rate of 0.01 for the Adam optimizer. For the other optimizers, we only show results for a learning rate of 0.01 which fares better than their default rate of 0.001. The learning curve corresponding to the plain vanilla gradient descent using the derived bound is also included for the comparison.

The proposed binary search algorithm outperforms all the other methods in all the cases as illustrated in Fig. 5. (Kindly note that the legend provided in the first subplot holds for the all the figures and is not repeated for ease of viewing.) It should be noted that all the above methods (Adam, RMSProp, Adagrad, Adadelta and gradient descent with derived bound) only require a single evaluation of the optimization algorithm whereas the BinarySearch method is employed when multiple evaluations can be performed; for our experiments, we have considered 10 evaluations for the binary search method. However, we can see that the performance of the optimization algorithm Adam with its default learning rate is fairly poor as compared to the tuned version. Note that this tuning would also take up evaluations based on the search algorithm employed. In the case of ReLU activation, we note that the derived bound itself outperforms all the other optimization methods. In case of the sigmoid activation function, the Adam optimizer with a learning rate of 0.01 exhibits faster convergence than the derived bound for $d = 10$ and $k = 10$; both Adam and RMSProp with 0.01 outperform the derived bound for $d = 20$ and $k = 20$. However, the choice of learning rate as 0.01 would require some tuning as the default rate is 0.001.

B. Two hidden layer networks

1) Comparison with HyperOpt: For a two hidden layer network, we run GD for $T = 200$ epochs as it takes greater number of epochs to converge than the single hidden layer. Similar to the case of a single hidden layer, the fraction of experiments for which HyperOpt chooses divergent values for a network with ReLU activation is tabulated in Table IV. We note that as the dimensions of the problem gets bigger,

			No. of evaluations		
d	k_1	k_2	5	10	20
5	3	2	0.07	0.02	0
10	5	3	0.29	0.23	0.05
5	10	5	0.44	0.34	0.21
10	10	10	0.77	0.5	0.44

TABLE IV: Fraction of times HyperOpt diverges for 2 hidden layer ReLU

the number of experiments which return unusable (divergent) learning rates increases. For example, in the case of ReLU activation with $k_1 = 10, k_2 = 10$, we obtain divergent learning rates for 44% of the experiments even after allowing 20 evaluations for HyperOpt. In case of sigmoid activation, it is again noted that there is no divergent behaviour. The fraction of the remaining experiments in which BinarySearch outperforms HyperOpt is tabulated in Table V. It is noticed that the proposed method overtakes the existing method at higher dimensions. Best-trace graphs for a two-hidden layer network resembles the graphs for a single hidden layer and are not produced due to lack of space.

			ReLU activation			Sigmoid activation		
			No. of evaluations			No. of evaluations		
d	k_1	k_2	5	10	20	5	10	20
5	3	2	0.45	0.65	0.59	1	0.98	0.95
10	5	3	0.58	0.67	0.61	1	0.99	0.96
5	10	5	0.52	0.62	0.73	0.4	0.93	0.96
10	10	10	0.56	0.66	0.86	0	1	1

TABLE V: Fraction of times the best value for BinarySearch outperforms HyperOpt for 2 hidden layer out of successful experiments

For a two-layer network, we consider the architecture with $d = 5, k_1 = 3, k_2 = 2$. This is chosen at random for study, and the constants over 100 experiments are tabulated in Table VI; we notice similar trends for other architecture with different widths as well.

Activation	Evals	Algorithm	Mean	Std. dev	Max	Min
ReLU	5	BinarySearch	0.431	0.435	3.287	0.102
		HyperOpt	0.285	0.239	0.996	0.015
	20	BinarySearch	0.424	0.391	2.952	0.061
		HyperOpt	0.308	0.218	0.915	0.001
Sigmoid	5	BinarySearch	6.012	0.929	8.055	3.983
		HyperOpt	0.677	0.222	0.999	0.122
	20	BinarySearch	18.656	6.662	32.364	5.849
		HyperOpt	0.756	0.177	0.991	0.136

TABLE VI: Variation of the chosen learning rates for two hidden layer network

From Table VI, we see that the average as well as the maximum learning rate chosen by BinarySearch is greater than HyperOpt. In our experiments, note that greater learning rate implies better convergence as we only consider non-divergent traces. Similar to the single hidden layer network,

(a) ReLU activation: d=10, k=10

(b) ReLU activation: d=20, k=20

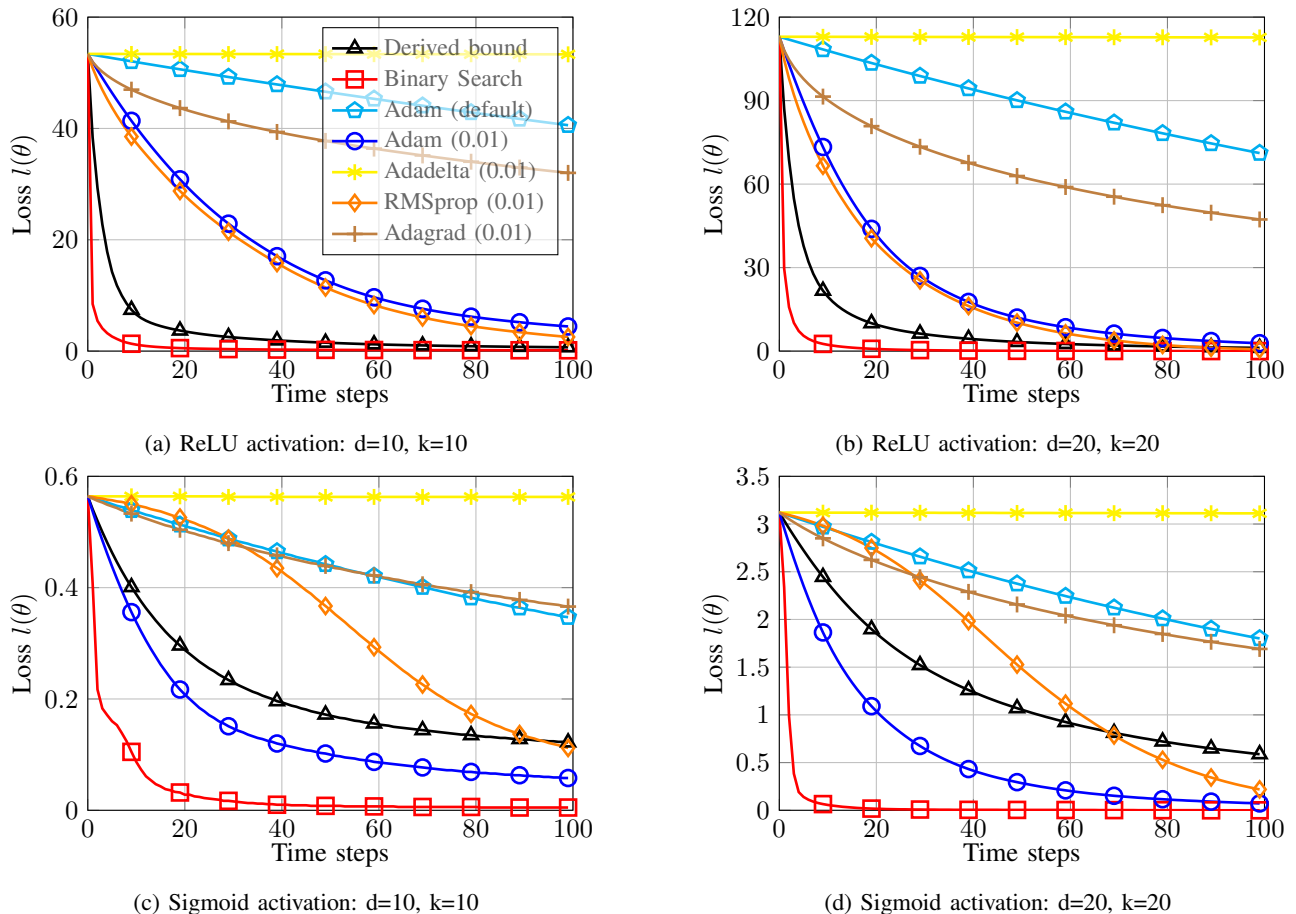(c) Sigmoid activation: d=10, k=10

(d) Sigmoid activation: d=20, k=20

Fig. 5: Single hidden layer network: Comparison with other optimization techniques

BinarySearch outperforms HyperOpt by a large margin in case of sigmoid activation, as it opts for learning rates greater than one.

2) Comparison with other optimization algorithms: For the case of two layer networks, we see that the proposed binary search (using 10 evaluations) always results in faster convergence as compared to the other optimization algorithms. This is shown in Fig. 6. Similar to that of a single hidden layer network, for ReLU activation, we see that the derived bound outperforms the other optimization methods. However, for sigmoid activation, especially in higher network dimensions, the other algorithms perform better than the derived bound but worse than the proposed binary search. Although all the above experiments were performed with $N = 100$, the trend in performance does not change with change in N as both the loss function as well as the derived bound contain a factor of $1/N$.

C. Remarks

1) Complexity of the algorithm: The complexity of the proposed method as well as the comparative method, namely HyperOpt, is $n\mathcal{C}(GD)$ where n is the number of evaluations and $\mathcal{C}(GD)$ is the complexity of the plain vanilla gradient descent algorithm for a fixed number of epochs. Note that both these tuning methods run the gradient descent algorithm during each evaluation for the same number of epochs. Therefore,

both algorithms have the same complexity as long as they employ the same number of evaluations. Newer optimization methods such as Adam, Adagrad, RMSProp and Adadelta have greater algorithmic complexity than the traditional gradient descent algorithm as they involve more additive and multiplicative operations in order to maintain an adaptive step-size. Although the difference in computational complexities is not much among the adaptive algorithms, the trend in complexity is as follows: GD < Adagrad < RMSProp < Adam. [40].

2) Using the derived bound in a different search: One could ask if the derived bound can be used in HyperOpt or other existing popular hyper-parameter optimization algorithms itself. Though we can employ the derived bounds to restrict the search space of existing algorithms on one end of the interval, how to define the other end is still a question. For example, we note that for a neural network with sigmoid activation function, GD converges for learning rates greater than 1. Hence, the learning rate corresponding to the derived bound (> 1) may be set as the lower limit of the search interval; however, how to set the upper bound still remains a question. We believe that this is worth exploring in future work.

VII. APPLICATIONS

Feedforward shallow networks are widely used in the context of resource allocation [41], wireless communication [22],

(a) ReLU activation: d=5, $k_1 = 3$, $k_2 = 2$

(b) ReLU activation: d=10, $k_1 = 10$, $k_2 = 10$

(c) Sigmoid activation: d=5, $k_1 = 3$, $k_2 = 2$

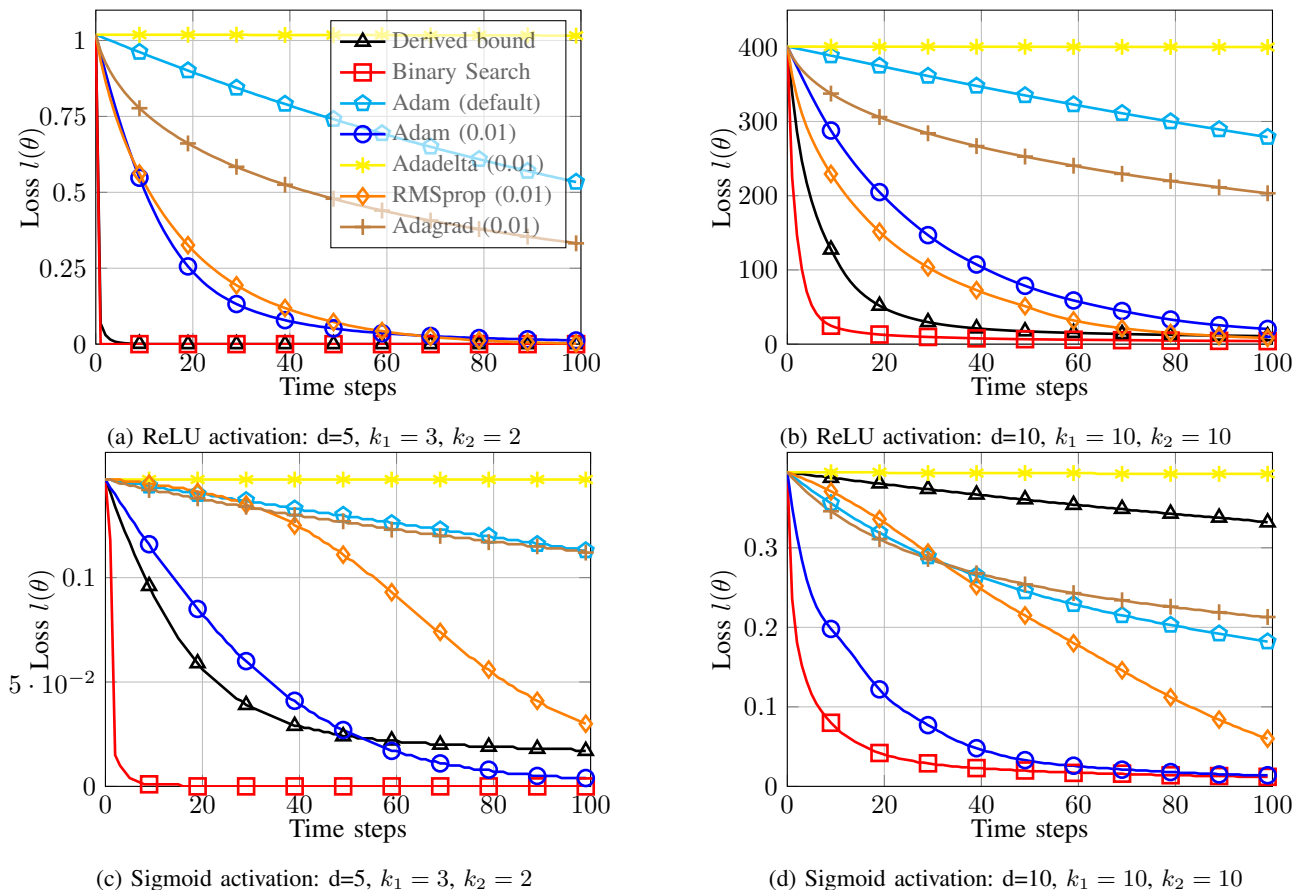(d) Sigmoid activation: d=10, $k_1 = 10$, $k_2 = 10$

Fig. 6: Two hidden layer network: Comparison with other optimization techniques

[24], financial predictions [23] and weather forecasting [42]. In this section, we illustrate the utility of the proposed algorithm in three specific applications.

A. Channel Estimation in OFDM systems

The use of neural network for channel estimation is advocated as traditional estimation methods such as Least Squares and MMSE suffer from lack of accuracy and high computational complexity respectively [21], [22]. We now describe the architecture employed in [22]. A pilot-based channel estimation is considered. A single hidden layer with k neurons with sigmoid activation function is employed. The real and the imaginary parts of the received pilots are fed separately into the network and the corresponding channel impulses are estimated at the output. The output layer (with linear activation) has the same number of neurons as the input layer, say $2M$ for estimating the channel response the real and imaginary parts of M sub-carriers. The component-wise sum of the squared difference between the estimated and actual channel response is the objective function to be minimized. The learning rate employed in the paper is 0.05 and is chosen through manual tuning, which usually involves searching through trial and error which is a laborious process. We now derive an upper bound on the gradient Lipschitz constant of the objective and apply Algorithm 1 to find the learning rate.

We follow the notation introduced in Section IV where the weight matrix between the input and hidden layer is denoted by $\boldsymbol{V}$ and the weight matrix between the hidden and output layer is denoted as $\boldsymbol{W}$. Let the data points be denoted as $(\boldsymbol{x}(i), \boldsymbol{o}(i))$ for $i = 1, ...N$. Each element of the output vector is denoted as $o(i)_{l_2}$ where $l_2 = 1, ..2M$. The loss function is given by,

$$l(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{l_2=1}^{2M} \left[\left(\sum_{l_1=1}^{k} \sigma(\boldsymbol{x}(i)^{\boldsymbol{T}} \boldsymbol{V}^{\boldsymbol{l_1}}) W_{l_1 l_2} \right) - o(i)_{l_2} \right]^2 . \tag{41}$$

Note that, in this application, the architecture consists of multiple outputs nodes. Therefore, the result in Theorem 1 cannot be used as it is. The bound on the gradient Lipschitz constant hence is derived for this specific case, and the bound is given by,

$$\alpha^* \leq \frac{1}{N} \sum_{i=1}^{N} \left[\frac{k_1 k_2}{16} \beta^2 \|\boldsymbol{x}(i)\|_\infty \|\boldsymbol{x}(i)\|_1 + \right.$$
$$\sum_{l_2=1}^{k_2} \left[(k_1 \beta - o_{l_2}) \frac{\beta}{10} \|\boldsymbol{x}(i)\|_\infty \|\boldsymbol{x}(i)\|_1 \right] +$$
$$\left. \frac{k_1 k_2}{4} \beta \|\boldsymbol{x}(i)\|_\infty + \frac{k_1}{4} (k_1 \beta - \min_m o_m) \|\boldsymbol{x}(i)\|_\infty \right]. \tag{42}$$

Sketch of Proof: The elements of the Hessian matrix $\nabla^2 l(\boldsymbol{\theta})$ are computed and the Gershgorin theorem (Theorem 3) is then applied to obtain the above result.

Here, we consider a OFDM system with $M = 64$ sub-carriers where all the sub-carriers consist of the pilot symbol. The pilots are transmitted through the channel and received. All the simulations are performed in the frequency domain. It is assumed that the channel impulse responses are available for training. As done in [22], the number of inputs and outputs to the neural network are $2M$ and the number of neurons in the hidden layer are $k = 10$. For our simulations, we have considered a QPSK constellation and an SNR of 10dB.

The learning rate chosen in the paper is a fixed learning rate 0.05. The loss corresponding to the fixed learning rate after $T = 100$ time steps is 0.068. The learning rate chosen by Algorithm 1 and the corresponding loss in tabulated in Table VII. We can see that the proposed method finds a learning

	No. of evaluations		
	5	10	20
Learning rate	0.033	0.062	0.064
Loss	0.0004	3.53e-6	3.27 e-23

TABLE VII: Learning rates chosen and loss encountered for channel estimation by Algorithm 1

rate that is comparable to the one suggested by manual tuning with as low as 5 evaluations. We can also see that the loss that the algorithm converges to is lower than the loss arrived at by using 0.05 as the learning rate.

B. Exchange rate prediction

Neural networks are used in various aspects of finance such as debt risk assessment, currency prediction, business risk failure, etc. [43]. Applications such as exchange rate prediction hold great importance in the economy. In [23], a single hidden layer neural network is considered where the neurons employ the sigmoid activation function. In the mentioned work, prediction is done using daily, monthly or quarterly steps. For the sake of our demonstration, we consider the daily step prediction. The exchange rates for the previous $d = 5$ days are fed as the input to the network and the prediction for the next day is made. The architecture of the network is the same as the one demonstrated in Fig. 1 with $d = 5$ input neurons, $k = 10$ neurons at the hidden layer and one output neuron. The data for the experiment is obtained from the website http://www.global-view.com/forex-trading-tools/forex-history/index.html as in [23].

The data is organized as $(\boldsymbol{x}(i), y(i))$ for $i = 1, ..N$ training samples; note that $\boldsymbol{x}(i) \in \mathbb{R}^d$ represents the daily step (change in the exchange rate from the previous day) for the past five days and $y(i)$ is the rate for the day (which is the quantity to be estimated). We implement [23] with a slight modification: the network proposed in the paper uses a threshold within every neuron which is also a parameter to be tuned; instead, in this implementation, we add a column of ones to the data to compensate for threshold. Hence, we have $\boldsymbol{x}(i) \in \mathbb{R}^{d+1}$. We are justified in doing so as we would tune the weight vector corresponding to the $d + 1$th input to the hidden layer instead of tuning the threshold. The loss function in [23] is

given by,

$$l(\boldsymbol{w}) = \frac{1}{2N} \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{k} \sigma(\boldsymbol{x}(i)^T \boldsymbol{w}^j) \right) - y(i) \right)^2, \quad (43)$$

where $\boldsymbol{w}$ denotes the weights of the network to be optimized and $\sigma(.)$ denotes the sigmoid activation function. We note that the loss function is the similar to (4) and hence the bound derived in (25) in Section III can be used.

The paper recommends GD as the optimization algorithm to be used; however, it does not recommend any tuning method for the learning rate for this application. We employ the BinarySearch method proposed in Algorithm 1 and tabulate the losses encountered after tuning the learning rate for $T = 500$ time steps in Table VIII. We note that the proposed method

	No. of evaluations		
	5	10	20
BinarySearch	0.254	0.2532	0.253
HyperOpt	0.255	0.255	0.254

TABLE VIII: Loss encountered for exchange rate prediction by Algorithm 1 and HyperOpt

performs well as compared to HyperOpt using TPE and is able to achieve the optimal loss within a small number of iterations. As it is noted that both the algorithms converge to similar losses, we wish to demonstrate the convergence graphs by plotting the best-trace graphs. From Fig. 7, we note that the proposed BinarySearch algorithm converges faster.
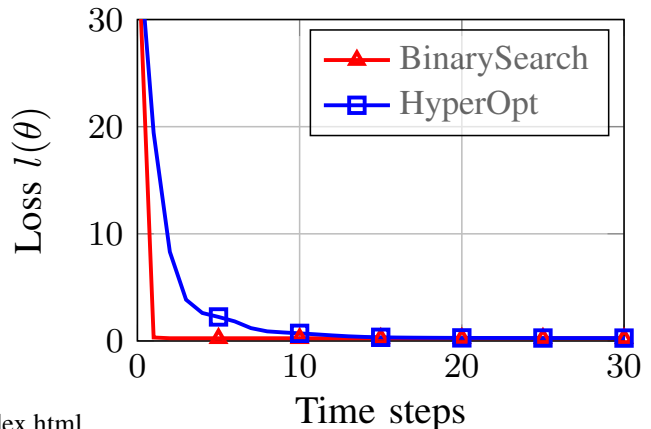

Fig. 7: Best trace comparison for exchange rate prediction network with 10 evaluations

C. Offset estimation in OFDM receivers

Recent work in [44] uses neural network blocks for different purposes while designing an OFDM receiver such as synchronization. We focus on the estimation of the Carrier Frequency Offset(CFO). Shallow networks with restricted width are employed to reduce the computational complexity.

Simulation setup of [44] is used and is briefly described below. The OFDM signal is generated as per IEEE 802.11 standard

using 64 subcarriers and 4-equally spaced pilots where 16-QAM constellation is employed for modulation. The baseline model is derived using the estimate from the cyclic prefix (CP). A moving window of the CP estimates derived from N_{CFO} consecutive OFDM symbols serve as inputs to the shallow neural network. The estimate from the established preamble method developed by Moose [45] is used as the label for training.

The said shallow network is constructed with the same architecture in Fig.1 with ReLU activation; hence, the upper bound on the gradient Lipschitz constant is given in (32). It is established in [44] that using a neural network to estimate the CFO as mentioned above results in better MSE than simply using the CP estimates or the preamble methods. We only verify if the proposed tuning method results in better learning curves than the optimization algorithm used in [44].

The authors employ the Adam optimization algorithm with the default learning rate of 0.001. We now compare this with the proposed method in Fig. 8. It can be seen from the figure that using the derived bound as the learning rate of GD (as well as the binary search method) converges within a few initial epochs whereas Adam optimizer takes 800 epochs to converge to the same minimum. Note that although binary search requires more evaluations, GD with the derived learning rate requires just one iteration, like the Adam optimizer. This shows that our proposed method results in faster convergence.
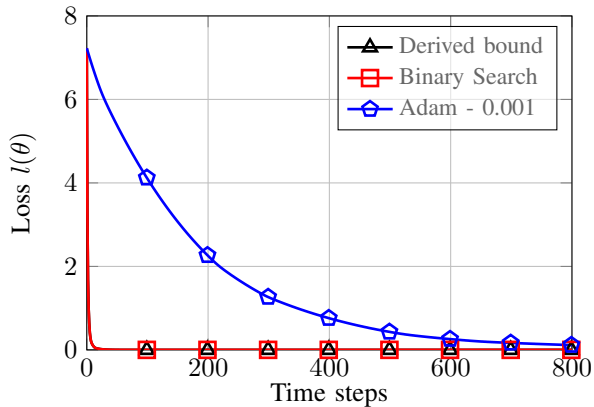

Fig. 8: Learning curves for CFO estimation

In this section, we considered three popular applications in the communication and finance sector where shallow feedforward networks are used and demonstrated that the proposed method can be used effectively to tune the learning rate as compared to the state-of-the-art tuning algorithms.

VIII. CONCLUDING REMARKS

In this work, we proposed a theory-based approach for determining the learning rate for a shallow feedforward neural network. We derived the gradient Lipschitz constant for fixed architectures and developed a search algorithm that employs the derived bound to find a better learning rate while ensuring convergence. While the existing algorithms tune harder, i.e., employ higher number of evaluations in order to find a suitable

learning rate, we can tune smarter by searching over an interval which is customized to the objective. When allowed the same number of evaluations, we demonstrated that the proposed method outperforms state-of-the-art methods such as HyperOpt in terms of convergence in both synthetic and real data.

APPENDIX A
PROOF OF THEOREM 2

As the function is doubly differentiable, the required constant is $\alpha^* = \max_{\boldsymbol{w}} \lambda_{max}(\nabla^2 l(\boldsymbol{w}))$.

$$\nabla l(\boldsymbol{w}) = \left(\sum_{j=1}^{k} s(\boldsymbol{x}^T \boldsymbol{w}^j) - y \right) \begin{bmatrix} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^1 \geq 0\}} \boldsymbol{x} \\ \vdots \\ \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^k \geq 0\}} \boldsymbol{x} \end{bmatrix} \quad (44)$$

$$\nabla^2 l(\boldsymbol{w}) = \begin{bmatrix} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^1 \geq 0\}} \boldsymbol{x} \\ \vdots \\ \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^k \geq 0\}} \boldsymbol{x} \end{bmatrix} \begin{bmatrix} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^1 \geq 0\}} \boldsymbol{x} \\ \vdots \\ \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^k \geq 0\}} \boldsymbol{x} \end{bmatrix}^T$$
$$= \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})^T \quad (45)$$

where

$$\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) \triangleq \begin{bmatrix} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^1 \geq 0\}} \boldsymbol{x} & \cdots & \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{w}^k \geq 0\}} \boldsymbol{x} \end{bmatrix}^T. \quad (46)$$

Although the ReLU function given by $\max(0, x)$ is non-differentiable at $x = 0$, the work in [20] states that if the input is assumed to be from the Gaussian distribution, the loss function becomes smooth, and the gradient is well defined everywhere. The gradient is given by $\mathbb{I}_{\{x \geq 0\}}$ where $\mathbb{I}$ is the indicator function. By a similar argument, we consider the second derivative to be zero over the entire real line. Note that the Gaussian assumption is only to ensure that the derivative of the ReLU function is defined at $x = 0$ due to the smoothness for theoretical tractability. The gradient Lipschitz constant is given by

$$\alpha^* = \max_{\boldsymbol{w}} \lambda_{max}(\nabla^2 l(\boldsymbol{w})) = \max_{\boldsymbol{w}} \lambda_{\max}(\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})^T). \quad (47)$$

We note that $\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})^T$ is a rank-1 matrix and therefore, its only non-zero eigenvalue is given by $\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})^T \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w}) = \|\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})\|^2$, which is also the maximum eigenvalue. Substituting in (47),

$$\alpha^* = \max_{\boldsymbol{w}} \|\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{w})\|^2. \quad (48)$$

The norm is maximized when all the entries of the vector are non-zero, i.e., when all the indicators correspond to 1. Let us define

$$\bar{\boldsymbol{a}}(\boldsymbol{x}) \triangleq \begin{bmatrix} \boldsymbol{x} & \cdots & \boldsymbol{x} \end{bmatrix}^T, \quad (49)$$

which is a stack of the input vector repeated k times. Therefore, the required constant is given by

$$\alpha^* = \|\bar{\boldsymbol{a}}(\boldsymbol{x})\|^2 = k \|\boldsymbol{x}\|^2. \quad (50)$$

In this case, we note that the derived constant for a single data point is not a bound, but the exact gradient Lipschitz constant and it is a function of the number of neurons, k, and the norm of the input vector.

APPENDIX B
PROOF OF LEMMA 3

The Rayleigh quotient of a Hermitian matrix $\boldsymbol{A}$ and a non-zero vector $\boldsymbol{g}$ is given by $\frac{\boldsymbol{g}^T \boldsymbol{A} \boldsymbol{g}}{\boldsymbol{g}^T \boldsymbol{g}}$ and reaches the maximum eigenvalue when the vector $\boldsymbol{g}$ is the eigen vector corresponding to the maximum eigenvalue [46].

$$\lambda_{max}(\boldsymbol{A}) = \max_{\boldsymbol{g}:\|\boldsymbol{g}\|=1} \boldsymbol{g}^T \boldsymbol{A} \boldsymbol{g}, \tag{51}$$

Also, observe that for any other vector of unit norm $\boldsymbol{h} \neq \boldsymbol{g}$,

$$\boldsymbol{g}^T \boldsymbol{A} \boldsymbol{g} > \boldsymbol{h}^T \boldsymbol{A} \boldsymbol{h}. \tag{52}$$

In the following proof, denoting $\boldsymbol{x}(i)$ as $\boldsymbol{x}_i$ and the principal eigen vectors of $\left(\sum_{i=1}^N \bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right)$, $\left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right)$ and $\left(\sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right)$ as $\bar{\boldsymbol{g}}, \boldsymbol{g}_i$ and $\hat{\boldsymbol{g}}$ respectively,

$$\lambda_{max}\left(\sum_{i=1}^N \bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) = \bar{\boldsymbol{g}}^T \left(\sum_{i=1}^N \bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \bar{\boldsymbol{g}}$$

$$= \sum_{i=1}^N \bar{\boldsymbol{g}}^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \bar{\boldsymbol{g}}$$

$$\geq \sum_{i=1}^N \boldsymbol{g}_i^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \boldsymbol{g}_i^T.$$

Note that as $\left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right)$ is a rank-1 matrix, the principal eigen vector is given by $\boldsymbol{g}_i = \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})$. Hence,

$$\sum_{i=1}^N \boldsymbol{g}_i^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \boldsymbol{g}_i^T$$

$$= \sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\right) \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w}). \tag{53}$$

Considering each term in the summation,

$$\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})$$

$$= \left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \bar{\boldsymbol{a}}(\boldsymbol{x}_i)\right)\left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\right)$$

$$= \left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i^T \boldsymbol{x}_i\right)\left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i^T \boldsymbol{x}_i\right)$$

$$= \left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}}^2 \boldsymbol{x}_i^T \boldsymbol{x}_i\right)\left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}}^2 \boldsymbol{x}_i^T \boldsymbol{x}_i\right)$$

$$= \left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i^T \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i\right)$$

$$\left(\sum_{j=1}^k \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i^T \mathbb{I}_{\{\boldsymbol{x}_i^T \boldsymbol{w}^j \geq 0\}} \boldsymbol{x}_i\right)$$

$$= \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right) \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w}).$$

Using this result in (53),

$$\sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \left(\bar{\boldsymbol{a}}(\boldsymbol{x}_i)\bar{\boldsymbol{a}}(\boldsymbol{x}_i)^T\right) \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})$$

$$= \sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T \left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right) \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})$$

$$\geq \sum_{i=1}^N \hat{\boldsymbol{g}}^T \left(\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right) \hat{\boldsymbol{g}}$$

$$= \hat{\boldsymbol{g}}^T \left(\sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right) \hat{\boldsymbol{g}}$$

$$= \lambda_{max}\left(\sum_{i=1}^N \boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})\boldsymbol{a}(\boldsymbol{x}_i,\boldsymbol{w})^T\right).$$

Hence proved.

APPENDIX C
PROOF OF THEOREM 5

The loss function is doubly differentiable, and hence,

$$\alpha^* = \max_{\boldsymbol{\theta}} \lambda_{max}(\nabla^2 l(\boldsymbol{\theta})). \tag{54}$$

The first-order partial derivatives are computed as follows,

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = A \frac{\partial A}{\partial \boldsymbol{\theta}} \tag{55}$$

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \left(\sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^1)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^1))W_{1l_2}\boldsymbol{x}\right]\right) \\ \vdots \\ \left(\sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{k_1})(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^{k_1}))W_{k_1l_2}\boldsymbol{x}\right]\right) \\ q_1\sigma(\boldsymbol{x}^T\boldsymbol{V}^1) \\ \vdots \\ q_1\sigma(\boldsymbol{x}^T\boldsymbol{V}^{k_1}) \\ q_2\sigma(\boldsymbol{x}^T\boldsymbol{V}^1) \\ \vdots \\ q_2\sigma(\boldsymbol{x}^T\boldsymbol{V}^{k_1}) \\ \vdots \\ q_{k_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{k_1}) \end{bmatrix} \tag{56}$$

We define the following terms, where q_a is the first derivative and q_a' is the second derivative of $\sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)$ and then compute the elements of the Hessian matrix.

$$A \triangleq \left(\sum_{l_2=1}^{k_2}\sigma\left(\sum_{l_1=1}^{k_1}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1l_2}\right) - y\right), \tag{57}$$

$$q_a \triangleq \sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)$$

$$\left(1 - \sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)\right), \tag{58}$$

$$q'_a \triangleq \sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)$$
$$\left(1-\sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)\right)$$
$$\left(1-2\sigma\left(\sum_{l_1=1}^{k_1}(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{l_1})W_{l_1a})\right)\right). \tag{59}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij}\partial W_{i'j'}} = \left(q_j\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)\right)\left(q_{j'}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})\right)$$
$$+ A\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})q'_j\mathbb{I}_{\{j=j'\}} \tag{60}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i\partial \boldsymbol{V}^{i'}} = \left(\sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))W_{il_2}\boldsymbol{x}\right]\right)$$
$$\left(\sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))W_{i'l_2}\boldsymbol{x}\right]\right)^T$$
$$+ A\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})$$
$$(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))\left[\sum_{l_2=1}^{k_2}q'_{l_2}W_{il_2}W_{i'l_2}\right]\boldsymbol{x}\boldsymbol{x}^T$$
$$+ A\left[\sum_{l_2=1}^{k_2}q_{l_2}W_{il_2}\right]\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))$$
$$(1-2\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))\mathbb{I}_{\{i=i'\}}\boldsymbol{x}\boldsymbol{x}^T \tag{61}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i\partial W_{i'j'}} = \sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))W_{il_2}\right]$$
$$q'_j\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})\boldsymbol{x}^T + A\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))$$
$$\left[\sum_{l_2=1}^{k_2}(q_{l_2}\mathbb{I}_{\{i=i',l_2=j'\}} + W_{il_2}q'_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))\right]\boldsymbol{x}^T \tag{62}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij}\partial \boldsymbol{V}^{i'}} = q_j\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)$$
$$\sum_{l_2=1}^{k_2}\left[q_{l_2}\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'})(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))W_{i'l_2}\boldsymbol{x}\right]$$
$$+ Aq_j\sigma(\boldsymbol{x}^T\boldsymbol{V}^i)(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^i))\mathbb{I}_{i=i'}\boldsymbol{x}$$
$$+ A(\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))^2(1-\sigma(\boldsymbol{x}^T\boldsymbol{V}^{i'}))q'_jW_{i'j}\boldsymbol{x} \tag{63}$$

It is observed that the elements of the Hessian matrix depends on the values of the parameters in $\boldsymbol{\theta}$ (through A) unlike the case with a single hidden layer in which the parameters only appeared as indicators. As the maximization is over $\boldsymbol{\theta}$, the elements of the matrix $\boldsymbol{V}$ and $\boldsymbol{W}$ can be scaled up arbitrarily and the obtained upper bound will be infinity, which is a trivial upper bound. To avoid this, it is assumed that the magnitude of the weights are restricted; i.e., $|\theta_i| < \beta \quad \forall i$. The Hessian matrix can be written in the following form:

$$\nabla^2 l(\boldsymbol{\theta}) = \left(\frac{dA}{d\boldsymbol{\theta}}\right)\left(\frac{dA}{d\boldsymbol{\theta}}\right)^T + \boldsymbol{M}, \tag{64}$$

where the first terms in all the second order partial derivative elements (given in (60) - (63)) are accounted for in $\left(\frac{dA}{d\boldsymbol{\theta}}\right)\left(\frac{dA}{d\boldsymbol{\theta}}\right)^T$. The rest of the additive terms are represented by the matrix $\boldsymbol{M}$. Applying Weyl's inequality (i.e., (12)),

$$\lambda_{max}(\nabla^2 l(\boldsymbol{\theta})) \leq \lambda_{max}\left(\left(\frac{dA}{d\boldsymbol{\theta}}\right)\left(\frac{dA}{d\boldsymbol{\theta}}\right)^T\right) + \lambda_{max}(\boldsymbol{M}). \tag{65}$$

We note that the first term in the above equation is a rank one matrix and has a maximum eigenvalue of $\left\|\frac{dA}{d\boldsymbol{\theta}}\right\|^2$. The required gradient Lipschitz constant is obtained by maximizing (65) over all values of $\boldsymbol{\theta}$ and is given by

$$\max_{\boldsymbol{\theta}}\lambda_{max}(\nabla^2 l(\boldsymbol{\theta})) \leq \max_{\boldsymbol{\theta}}\left\|\frac{dA}{d\boldsymbol{\theta}}\right\|^2 + \max_{\boldsymbol{\theta}}\lambda_{max}(\boldsymbol{M}) \tag{66}$$

Focusing on the first term in (66), the vector $\frac{dA}{d\boldsymbol{\theta}}$ consists of k_1k_2 terms of the form $q_{(.)}\sigma(.)$ and k_1 terms of the form $\sum_{l_2=1}^{k_2}q_{l_2}\nabla\sigma(\boldsymbol{x}^T\boldsymbol{V}^a)W_{al_2}\boldsymbol{x}$ where $a=1,...,k_1$. Recall from (16) that $\sigma(.) \leq 1$ and from (17) that $q_{(.)} \leq \frac{1}{4}$. Therefore,

$$\max_{\boldsymbol{\theta}}\left\|\frac{dA}{d\boldsymbol{\theta}}\right\|^2 = k_1\left(\frac{k_2\beta\|\boldsymbol{x}\|}{16}\right)^2 + \frac{k_1k_2}{16} \tag{67}$$

where $\beta = \max_i\theta_i$. We now focus on the second additive term in (66). To bound the maximum eigenvalue of $\boldsymbol{M}$, the Gershgorin's theorem (stated in Theorem 3) is employed. Considering the terms in (60) - (63) that are not included in $\left(\frac{dA}{d\boldsymbol{\theta}}\right)\left(\frac{dA}{d\boldsymbol{\theta}}\right)^T$, we bound the maximum row sum over all possible values of $\boldsymbol{\theta}$. The row sum can be computed in one of two possible ways considering elements from (a) $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i\partial \boldsymbol{V}^{i'}}$ and $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i\partial W_{i'j'}}$ or (b) $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij}\partial \boldsymbol{V}^{i'}}$ and $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij}\partial W_{i'j'}}$. The maximum value taken by A is $|k_2 - y|$ as the sigmoid function has a maximum value of one. Recall that q_a is a first derivative and q'_a is a second derivative of the sigmoid function. Using the bounds on derivatives stated in (16) - (18),

$$\max_{\boldsymbol{\theta}}\lambda_{max}(\boldsymbol{M}) \leq |k_2 - y|\max\left(\frac{1}{10} + \left[\frac{1}{4} + \frac{\beta}{10}\right]\frac{k_1\|\boldsymbol{x}\|_1}{4},\right.$$
$$\left.\left[\frac{1}{4} + \frac{\beta}{10}\right]\frac{k_2\|\boldsymbol{x}\|_\infty}{4} + \left[\frac{\beta}{1000} + \frac{1}{4}\right]k_1k_2\beta\|\boldsymbol{x}\|_1\|\boldsymbol{x}\|_\infty\right) \tag{68}$$

where the first argument in the maximization corresponds to case (a) and the second argument corresponds to case (b) of computing the row sum. Combining (67) and (68),

$$\alpha^* \leq \max\left(\frac{1}{10} + \left[\frac{1}{4} + \frac{\beta}{10}\right]\frac{k_1\|\boldsymbol{x}\|_1}{4}, \left[\frac{1}{4} + \frac{\beta}{10}\right]\frac{k_2\|\boldsymbol{x}\|_\infty}{4}\right.$$
$$\left.+ \left[\frac{\beta}{1000} + \frac{1}{4}\right]k_1k_2\beta\|\boldsymbol{x}\|_1\|\boldsymbol{x}\|_\infty\right)|k_2 - y|$$
$$+ k_1\left(\frac{k_2\beta\|\boldsymbol{x}\|}{16}\right)^2 + \frac{k_1k_2}{16} \tag{69}$$

APPENDIX D
PROOF OF THEOREM 6

The aim is to find the gradient Lipschitz constant of $l(\boldsymbol{\theta})$. For a doubly differentiable function, the required constant is given by

$$\alpha^* = \max_{\boldsymbol{\theta}} \lambda_{max}(\nabla^2 l(\boldsymbol{\theta})). \tag{70}$$

In order to find the Hessian, we initially find the first-order partial derivatives:

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = A \frac{\partial A}{\partial \boldsymbol{\theta}} \tag{71}$$

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^1 \geq 0\}} W_{1 l_2} \boldsymbol{x} \right] \right) \\ \vdots \\ \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^{k_1} \geq 0\}} W_{k_1 l_2} \boldsymbol{x} \right] \right) \\ q_1 s(\boldsymbol{x}^T \boldsymbol{V}^1) \\ q_1 s(\boldsymbol{x}^T \boldsymbol{V}^2) \\ \vdots \\ q_1 s(\boldsymbol{x}^T \boldsymbol{V}^{k_1}) \\ \vdots \\ q_{k_2} s(\boldsymbol{x}^T \boldsymbol{V}^{k_1}) \end{bmatrix} \tag{72}$$

where

$$A \triangleq \left(\sum_{l_2=1}^{k_2} s \left(\sum_{l_1=1}^{k_1} s(\boldsymbol{x}^T \boldsymbol{V}^{l_1}) W_{l_1 l_2} \right) - y \right) \tag{73}$$

$$q_a \triangleq \mathbb{I}_{\{\sum_{l_1=1}^{k_1} (s(\boldsymbol{x}^T \boldsymbol{V}^{l_1}) W_{l_1 a}) \geq 0\}}. \tag{74}$$

Similar to one-hidden layer ReLU case, we assume that the gradients of q_a with respect to W_{ij} and $\boldsymbol{V}^i$ are 0 and $\boldsymbol{0}$ respectively. Now, the second-order partial derivatives are derived.

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij} \partial W_{i'j'}} = \left(q_j s(\boldsymbol{x}^T \boldsymbol{V}^i) \right) \left(q_{j'} s(\boldsymbol{x}^T \boldsymbol{V}^{i'}) \right) \tag{75}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i \partial \boldsymbol{V}^{i'}} = \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^i \geq 0\}} W_{i l_2} \boldsymbol{x} \right] \right) \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^{i'} \geq 0\}} W_{i' l_2} \boldsymbol{x} \right] \right)^T \tag{76}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^i \partial W_{i'j'}} = A q_{j'} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^{i'} \geq 0\}} \mathbb{I}_{\{i=i'\}} \boldsymbol{x}^T + \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^i \geq 0\}} W_{i l_2} \boldsymbol{x}^T \right] \right) \left(q_{j'} s(\boldsymbol{x}^T \boldsymbol{V}^{i'}) \right) \tag{77}$$

$$\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij} \partial \boldsymbol{V}^{i'}} = A q_j \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^i \geq 0\}} \mathbb{I}_{\{i=i'\}} \boldsymbol{x} + \left(\sum_{l_2=1}^{k_2} \left[q_{l_2} \mathbb{I}_{\{\boldsymbol{x}^T \boldsymbol{V}^{i'} \geq 0\}} W_{i' l_2} \boldsymbol{x} \right] \right) \left(q_j s(\boldsymbol{x}^T \boldsymbol{V}^i) \right). \tag{78}$$

Note that the Hessian is a square matrix of dimension $k_1(d + k_2) \times k_1(d + k_2)$. On putting the Hessian matrix together, it is observed that the Hessian can be written as a sum of two matrices as given below

$$\nabla^2 l(\boldsymbol{\theta}) = \left(\frac{dA}{d\boldsymbol{\theta}} \right) \left(\frac{dA}{d\boldsymbol{\theta}} \right)^T + \boldsymbol{M}, \tag{79}$$

where $\boldsymbol{M}$ is a matrix with all the elements as zero except for the additional elements corresponding to $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial W_{ij} \partial \boldsymbol{V}^{i'}}$ and $\frac{\partial^2 l(\boldsymbol{\theta})}{\partial \boldsymbol{V}^{i'} \partial W_{ij}}$ where $i = i'$. The main diagonal elements of the matrix are always zero and it is also symmetric; there are $2dk_1k_2$ non-zero elements in the matrix.

Using Weyl's inequality stated in (12),

$$\lambda_{max}(\nabla^2 l(\boldsymbol{\theta})) \leq \lambda_{max}\left(\left(\frac{dA}{d\boldsymbol{\theta}} \right) \left(\frac{dA}{d\boldsymbol{\theta}} \right)^T \right) + \lambda_{max}(\boldsymbol{M}) \tag{80}$$

$$= \left\| \frac{dA}{d\boldsymbol{\theta}} \right\|^2 + \lambda_{max}(\boldsymbol{M}). \tag{81}$$

The maximum eigenvalue of the matrix $\boldsymbol{M}$ can be bounded using the Brauer's Ovals of Cassini bound (stated in Theorem 4).

$$\lambda_{max}(\boldsymbol{M}) \leq \max_{i \neq j} \left(\frac{m_{ii} + m_{jj}}{2} + \sqrt{(m_{ii} - m_{jj})^2 + R_i(\boldsymbol{M}) R_j(\boldsymbol{M})} \right) \tag{82}$$

where $R_i(\boldsymbol{M}) = \sum_{i \neq j} |m_{ij}|$. It is noted that all diagonal elements are always zero and multiple rows have similar row sums. Therefore, the bound reduces to

$$\lambda_{max}(\boldsymbol{M}) \leq \max_i R_i(\boldsymbol{M}) \tag{83}$$

This is the same as the Gershgorin's bound obtained for the matrix $\boldsymbol{M}$. Note that the elements of the matrix $\boldsymbol{M}$ are the first terms in (77) and (78) corresponding to the case when $i = i'$. The structure of the matrix $\boldsymbol{M}$ is such that the maximum row sum can be computed in one of two ways: Ak_2 times the maximum element of vector $\boldsymbol{x}$, or A times the sum of elements of $\boldsymbol{x}$. Therefore, while maximizing over θ, the maximum row sum of $\boldsymbol{M}$ is given by

$$\max_i R_i(\boldsymbol{M}) = \max(Ak_2|\boldsymbol{x}|_\infty, A|\boldsymbol{x}|_1), \tag{84}$$

where $|\boldsymbol{x}|_\infty = \max_i \boldsymbol{x}_i$ and $|\boldsymbol{x}|_1 = \sum_i \boldsymbol{x}_i$.

We can write (81) as

$$\lambda_{max}(\nabla^2 l(\boldsymbol{\theta})) \leq \left\| \frac{dA}{d\boldsymbol{\theta}} \right\|^2 + \max(Ak_2|\boldsymbol{x}|_\infty, A|\boldsymbol{x}|_1). \tag{85}$$

To obtain the desired bound on the gradient Lipschitz constant, we maximize over all possible values of $\boldsymbol{\theta}$ to obtain,

$$\max_{\boldsymbol{\theta}} \lambda_{max}(\nabla^2 l(\boldsymbol{\theta})) \leq \max_{\boldsymbol{\theta}} \left\| \frac{dA}{d\boldsymbol{\theta}} \right\|^2 + \max_{\boldsymbol{\theta}} \lambda_{max}(\boldsymbol{M}). \tag{86}$$

The first term is an outer product of vectors (matrix of rank 1) and hence, the eigenvalue is given by their inner product. The vector $\frac{dA}{d\boldsymbol{\theta}}$ consists of $k_1(d + k_2)$ terms each with an indicator, an element from θ and the input vector. Recall that

to avoid arbitrary scaling of the derived bound, we impose the following restriction that $|\theta_i| \leq \beta \quad \forall i$. Therefore,

$$\max_{\boldsymbol{\theta}} \left\| \frac{dA}{d\boldsymbol{\theta}} \right\|^2 = k_1(d+k_2)\beta^2 \|\boldsymbol{x}\|^2 . \tag{87}$$

To maximize the second term in (86), we note that the scalar term A is a sum of $k_1 k_2$ combinations of product of two weight parameters with the data vector $\boldsymbol{x}$. The maximum value that the scalar A can take is denoted by $A_{max} = k_1 k_2 \beta^2 \|\boldsymbol{x}\| - y$. Therefore, the second term is maximized as

$$\max_{\boldsymbol{\theta}} \lambda_{max}(\boldsymbol{M}) = \max((A_{max}k_2|\boldsymbol{x}|_\infty, A_{max}|\boldsymbol{x}|_1), \tag{88}$$

where $A_{max} = k_1 k_2 \beta^2 \|\boldsymbol{x}\| - y$. Combining (86), (87) and (88), we obtain

$$\alpha^* \leq k_1(d+k_2)\beta^2 \|\boldsymbol{x}\|^2 + \max((A_{max}k_2|\boldsymbol{x}|_\infty, A_{max}|\boldsymbol{x}|_1). \tag{89}$$

An upper bound on the gradient Lipschitz constant for a two hidden layer ReLU network is derived.

REFERENCES

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[2] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2595–2621, 2018.

[3] V. Raj and S. Kalyani, "Backpropagating through the air: Deep learning at physical layer without channel models," *IEEE Communications Letters*, vol. 22, no. 11, pp. 2278–2281, 2018.

[4] ——, "Design of communication systems using deep learning: A variational inference perspective," *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[5] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[6] M. Yu and T.-S. Chang, "An adaptive step size for backpropagation using linear lower bounding functions," *IEEE transactions on signal processing*, vol. 43, no. 5, pp. 1243–1248, 1995.

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

[8] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning*. Springer, 2019, pp. 3–33.

[9] D. C. Montgomery, "Design and analysis of experiments. john wiley & sons," *Inc., New York*, vol. 1997, pp. 200–1, 2001.

[10] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[11] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[12] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[13] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[14] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," *Jmlr*, 2013.

[15] A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6724–6728.

[16] S. Kallummil and S. Kalyani, "Signal and noise statistics oblivious orthogonal matching pursuit," in *International Conference on Machine Learning*, 2018, pp. 2429–2438.

[17] K. Chaudhuri, Y. Freund, and D. J. Hsu, "A parameter-free hedging algorithm," in *Advances in neural information processing systems*, 2009, pp. 297–305.

[18] V. Menon and S. Kalyani, "Structured and unstructured outlier identification for robust pca: A fast parameter free algorithm," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2439–2452, 2019.

[19] X. Zhang, Y. Yu, L. Wang, and Q. Gu, "Learning one-hidden-layer relu networks via gradient descent," in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 1524–1534.

[20] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with relu activation," in *Advances in Neural Information Processing Systems*, 2017, pp. 597–607.

[21] R. Jiang, X. Wang, S. Cao, J. Zhao, and X. Li, "Deep neural networks for channel estimation in underwater acoustic ofdm systems," *IEEE Access*, vol. 7, pp. 23 579–23 594, 2019.

[22] N. Taşpinar and M. N. Seyman, "Back propagation neural network approach for channel estimation in ofdm system," in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*. IEEE, 2010, pp. 265–268.

[23] S. Galeshchuk, "Neural networks performance in exchange rate prediction," *Neurocomputing*, vol. 172, pp. 446–452, 2016.

[24] H. Huang, J. Yang, H. Huang, Y. Song, and G. Gui, "Deep learning for super-resolution channel estimation and doa estimation based massive mimo system," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8549–8560, 2018.

[25] X. Ye, X. Yin, X. Cai, A. P. Yuste, and H. Xu, "Neural-network-assisted ue localization using radio-channel fingerprints in lte networks," *Ieee Access*, vol. 5, pp. 12 071–12 087, 2017.

[26] Z. Chen, Z. He, K. Niu, and Y. Rong, "Neural network-based symbol detection in high-speed ofdm underwater acoustic communication," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2018, pp. 1–5.

[27] Y. Nesterov, "Introductory lectures on convex programming volume i: Basic course," *Lecture notes*, 1998.

[28] R. Balan, M. Singh, and D. Zou, "Lipschitz properties for deep convolutional networks," *arXiv preprint arXiv:1701.05217*, 2017.

[29] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: analysis and efficient estimation," in *Advances in Neural Information Processing Systems*, 2018, pp. 3835–3844.

[30] A. J. Laub, *Matrix analysis for scientists and engineers*. Siam, 2005, vol. 91.

[31] J. Schlessman, "Approximation of the sigmoid function and its derivative using a minimax approach," 2002.

[32] R. S. Varga, *Matrix iterative analysis*. Springer Science & Business Media, 2009, vol. 27.

[33] L. DeVille, "Optimizing gershgorin for symmetric matrices," *Linear Algebra and its Applications*, 2019.

[34] C. H. Davis, "The binary search algorithm," *American Documentation (pre-1986)*, vol. 20, no. 2, p. 167, 1969.

[35] C. Biernacki and J. Jacques, "Model-based clustering of multivariate ordinal data relying on a stochastic binary search algorithm," *Statistics and Computing*, vol. 26, no. 5, pp. 929–943, 2016.

[36] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[37] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[38] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[39] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6d- a separate, adaptive learning rate for each connection," *Slides of Lecture Neural Networks for Machine Learning*, 2012.

[40] H. Wang, J. Zhou, Y. Wang, J. Wei, W. Liu, C. Yu, and Z. Li, "Optimization algorithms of neural networks for traditional time-domain equalizer in optical communications," *Applied Sciences*, vol. 9, no. 18, p. 3907, 2019.

[41] M. Eisen, C. Zhang, L. F. Chamon, D. D. Lee, and A. Ribeiro, "Learning optimal resource allocations in wireless systems," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2775–2790, 2019.

[42] A. Yadav and K. Sahu, "Wind forecasting using artificial neural networks: a survey and taxonomy," *International Journal of Research In Science & Engineering*, vol. 3, 2017.

[43] G. A. Perez, "Applicability of neural networks in finance across different countries," *Journal of Applied Engineering (JOAE)*, vol. 5, no. 7, 2017.

[44] M. A. Ouameur, A. D. T. Lê, and D. Massicotte, "Model-aided distributed shallow learning for ofdm receiver in ieee 802.11 channel model," *Wireless Networks*, pp. 1–10, 2020.

[45] P. H. Moose, "A technique for orthogonal frequency division multiplexing frequency offset correction," *IEEE Transactions on communications*, vol. 42, no. 10, pp. 2908–2914, 1994.

[46] P. Van Mieghem, "A new type of lower bound for the largest eigenvalue of a symmetric matrix," *Linear Algebra and its Applications*, vol. 427, no. 1, pp. 119–129, 2007.