Proceedings of Machine Learning Research vol 144:1–12, 2021

Robust error bounds for quantised and pruned neural networks

Jiaqi Li JIAQI.LI@ST-HUGHS.OX.AC.UK
Ross Drummond ROSS.DRUMMOND@ENG.OX.AC.UK
Stephen R. Duncan STEPHEN.DUNCAN@ENG.OX.AC.UK
Department of Engineering Science, University of Oxford, 17 Parks Road, OX1 3PJ, Oxford, United Kingdom

Abstract

With the rise of smartphones and the internet-of-things, data is increasingly getting generated at the edge on local, personal devices. For privacy, latency and energy saving reasons, this shift is causing machine learning algorithms to move towards decentralisation with the data and algorithms stored, and even trained, locally on devices. The device hardware becomes the main bottleneck for model capability in this set-up, creating a need for slimmed down, more efficient neural networks. Neural network pruning and quantisation are two methods that have been developed for this, with both approaches demonstrating impressive results in reducing the computational cost without sacrificing significantly on model performance. However, the understanding behind these reduction methods remains underdeveloped. To address this issue, a semi-definite program is introduced to bound the worst-case error caused by pruning or quantising a neural network. The method can be applied to many neural network structures and nonlinear activation functions with the bounds holding robustly for all inputs in specified sets. It is hoped that the computed bounds will provide certainty to the performance of these algorithms when deployed on safety-critical systems.

Keywords: Robustness; quantised neural networks; error bounds.

1. Introduction

The way we generate data is rapidly changing in response to the rise of the internet-of-things and the widespread use of smartphones (Zhou et al., 2019). In terms of data sources, there is an accelerating shift away from big data, like e-commerce records, stored on mega-scale cloud data centres towards more localised data sources generated by users interacting with their personal devices (Zhang et al., 2019). This evolving data landscape is causing a rethink to how machine learning algorithms are trained and implemented.

The most intuitive way to train and implement these algorithms within this new data landscape remains *centralised learning* where the data is sent to a central data centre in the cloud where the algorithm is then trained and run (Zhou et al., 2019). This approach should give the best model performance and is a natural extension of existing algorithm training frameworks but has several flaws (Zhou et al., 2019; Gündüz et al., 2019). These include: (i) Concerns around user privacy since the data is stored in a data centre which could be owned by a third party, (ii) The increased latency of centralised algorithms in responding to new data, and (iii.) As the algorithms get ever larger in response to the growing availability of data, they are requiring significantly more compute power, memory storage and energy.

These flaws are inherent to centralised learning and motivate growing efforts to direct machine learning towards a more *decentralised* approach. With decentralised learning (Zhang et al., 2019; Zhou et al., 2019) (often known as *edge intelligence* (Zhou et al., 2019; Deng et al., 2020; Shi et al., 2020) and falling under the umbrella of *distributed learning* (Zhang et al., 2019) amongst other labels), there is a greater emphasis on implementing and even training the algorithms locally on devices. In this way, decentralised learning aims to generate smaller, more personalised algorithms

that can help circumnavigate the above-mentioned issues with centralised learning- albeit typically doing so at the expense of model performance (Zhang et al., 2019). Running the algorithms locally on the devices means that hardware constraints (e.g. memory availability, compute power and energy consumption) become important. Managing these constraints whilst still delivering reasonable model performance is one of the main bottlenecks that decentralised learning needs to overcome and has led to a push for slimmed-down, efficient algorithms (typically neural networks (NNs)- the class of model focussed on in this work). As such, there has been increased interest in pruned neural networks which sparsify the weights and biases (Blalock et al., 2020) and quantised neural networks (e.g. (Shin et al., 2016; Courbariaux et al., 2014; Sung et al., 2015)) which use a coarse fixed-point representation of the algorithm's parameters. However, a full and robust understanding of how modifications like network pruning and quantisation affect the neural networks outputs, that goes beyond sensitivity studies like (Shin et al., 2016) for instance, is as yet still not fully developed.

To address these issues, this paper introduces a formal framework to compute performance guarantees for neural networks implemented on low-cost and memory-limited hardware. Specifically, we address the problem of determining how the output of a neural network is corrupted by either quantising its parameters and/or data after storing them using fixed-point arithmetic or from pruning it. In other words, a method to compute bounds for quantised and pruned neural networks is introduced. The main contributions are:

- A semi-definite program (SDP) to bound the worst-case error from neural network pruning or from quantising its weights, biases and input using fixed-point arithmetic.
- The computed bounds hold robustly for all input data in specified sets and account for the neural network's nonlinearities.
- The neural network quantisation and pruning bounds are shown to be special cases of a more general problem, called the neural network *similarity problem*, which bounds the worst-case error of two different neural networks outputs for all inputs in specific sets.

The neural network *similarity problem* (Problem 1) connects to the notion of incremental stability in dynamical systems (Zames, 1966), with the second neural network of the problem being either the quantised or pruned version.

Related work: Many results have demonstrated the potential of both quantised and pruned neural networks to realise machine learning on limited hardware. For example, Gong et al. (2014) achieved a 16-24$\times$ network compression for the 1000-category classification on ImageNet with only a 1% loss of classification accuracy and Lin et al. (2016) showed no loss in accuracy by reducing a model trained on the CIFAR-10 benchmark by $> 20\%$. More extreme levels of quantisation have also been explored, for example binarised neural networks (where the weights and/or activation functions are restricted to be binary) have also demonstrated rapid compute speeds and large memory savings without significant accuracy losses (Hubara et al. (2016) and Rastegari et al. (2016)).

The quantisation error bounds presented in this work are closely related to the semi-definite programmes for certifying neural network robustness developed in Fazlyab et al. (2019); Raghunathan et al. (2018); Dathathri et al. (2020). These SDPs guarantee that small perturbations in the input will not lead to large variations in the NN output and can help prevent against adversarial attacks and make the NNs more reliable to out-of-sample data. Recently, there has been a focus on improving the scalability of these SDP robustness certificates (Dathathri et al., 2020) (since they scale

by $\mathcal{O}(n^6)$ in runtime and $\mathcal{O}(n^4)$ in memory requirements, where n is the number of neurons in the network (Dathathri et al., 2020)), a particularly exciting direction for this paper as it could help make the presented bounds applicable to larger NNs.

Acronyms, Notations and Preliminaries

Throughout, frequent reference is made to the extended version of this paper (Li et al., 2021) containing the appendices. $\mathbb{R}_+$, $\mathbb{R}_+^n$ and $\mathbb{R}_+^{n\times n}$ denote real non-negative numbers, non-negative real vectors of dimension n and real non-negative matrices of size $n \times n$ respectively. Positive diagonal matrices of size $n \times n$ are denoted $\mathbb{D}_+^n$. $\mathbb{Z}_+$ ($\mathbb{Z}_-$) is the set of positive (negative) integers. The set of natural numbers (non-negative integers) is $\mathbb{N}$. The matrix of zeros of dimension $n \times m$ is $\mathbf{0}_{n\times m}$ and $\mathbf{0}_n$ is the vector of zeros of dimension n. The matrix of ones of dimension $N \times M$ is $\mathbf{1}_{N\times M}$ and $\mathbf{1}_N$ is the vector of ones of dimension N. The identity matrix of dimension N is $\mathbb{I}_N$. The $blkdiag\{a\}$ function takes an array as input and returns a block diagonal matrix with elements of a on its main diagonal as ordered in a. Strict and elementwise LMIs are posed as $F(x) > 0$ while non-strict and elementwise LMIs are written as $F(x) \geqslant 0$. A positive (negative) definite matrix Ω is denoted $\Omega \succ (\prec)\ 0$, and a positive (negative) semi-definite Γ matrix is written $\Gamma \succcurlyeq (\preccurlyeq)\ 0$. The p-norm is displayed by $||\cdot||_p : \mathbb{R}^n \to \mathbb{R}_+, p \in \mathbb{N}$. Where acronyms are used, "NN" stands for "artificial neural network", "QC" stands for quadratic constraint, "LMI" stands for "linear matrix inequality", and "SDP" stands for "semi-definite programme".

2. Neural Networks: Representation in state-space form

Consider two functions $f_1(x_1) : \mathcal{X} \to \mathcal{F}$ and $f_2(x_2) : \mathcal{X} \to \mathcal{F}$ acting upon input data $x_1, x_2 \in \mathcal{X}$. These functions are assumed to be defined by feed-forward NNs composed of ℓ hidden layers, with the k^{th} layer composed of n_k neurons and the total number of neurons being $N = \sum_{k=1}^{l} n_k$. Both the first and second neural network ($i \in \{1, 2\}$) can be expanded out in a state-space-like form

$$x_i^0 = x_i, \tag{1a}$$
$$x_i^{k+1} = \phi(W_i^k x_i^k + b_i^k), \quad k = 0, 1, \ldots, \ell - 1, \tag{1b}$$
$$f_i(x_i) = W_i^\ell x_i^\ell + b_i^\ell. \tag{1c}$$

Here, for neural network $i \in \{1, 2\}$, $x_i^0 \in \mathcal{X}_i \subseteq \mathbb{R}^{n_x}$ represents the input data, $W_i^k \in \mathbb{R}^{n_{k+1}\times n_k}$ the weights, $b_i^k \in \mathbb{R}^{n_{k+1}}$ the biases, $k = 0, \ldots, \ell - 1$, defines the network layers, n_x and n_f are, respectively, the network's input and output dimensions and $\phi(\cdot)$ are the activation functions- typically a Rectified Linear Unit (ReLU), sigmoid or tanh- which in this paper can be any function satisfying some of the properties of (Li et al., 2021, Definition 4). The activation functions are taken to act elementwise on their vector arguments.

2.1. Compact neural network representation

For the $i^{\text{th}} \in \{1,\, 2\}$ neural network, define the arguments of the activation functions $\phi(\cdot)$ as

$$\xi_i = \begin{bmatrix} \xi_i^1 \\ \xi_i^2 \\ \vdots \\ \xi_i^\ell \end{bmatrix} = \begin{bmatrix} W_i^0 & \mathbf{0}_{n_1 \times n_1} & \cdots & \mathbf{0}_{n_1 \times n_{l-1}} \\ \mathbf{0}_{n_2 \times n_2} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0}_{n_{\ell-1} \times n_{\ell-2}} \\ \mathbf{0}_{n_l \times n_x} & \cdots & \mathbf{0}_{n_1 \times n_{l-2}} & W_i^{\ell-1} \end{bmatrix} \begin{bmatrix} x_i^0 \\ x_i^1 \\ \vdots \\ x_i^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_i^0 \\ b_i^1 \\ \vdots \\ b_i^{\ell-1} \end{bmatrix} \in \mathbb{R}^M. \quad (2)$$

These vectors allow the i^{th} NN of (1) be written in the more compact form

$$\xi_i^{k+1} = W_i^k x_i + b_i^k, \quad k = 0, 1, \ldots, \ell - 1, \quad (3a)$$

$$x_i^{k+1} = \phi(\xi_i^{k+1}), \quad k = 0, 1, \ldots, \ell - 1, \quad (3b)$$

$$f_i(x_i) = W_i^\ell \phi(\xi_i^\ell) + b_i^\ell. \quad (3c)$$

The vectors

$$\mu = \left[{\xi_1}^T, \;\; {\xi_2}^T, \;\; \phi(\xi_1)^T, \;\; \phi(\xi_2)^T, \;\; 1\right]^T, \eta = \left[{x_1}^T, \;\; {x_2}^T, \;\; \phi(\xi_1)^T, \;\; \phi(\xi_2)^T, \;\; 1\right]^T, \quad (4)$$

will also be used throughout the paper to build the various inequalities. These two vectors are linearly related through a matrix $E \in \mathbb{R}^{2(n_x+M)+1 \times 4M+1}$ by $\eta = E\mu$.

2.2. The neural network similarity problem: Bounding the worst-case error between two neural networks

The neural network *similarity problem* is now introduced to bound the worst-case error between the outputs of two neural networks for all input data in pre-defined sets (e.g. hypercubes). Solutions to this problem indicate how similar two NNs are to each other.

Problem 1 *Given the two neural networks $f_1(x_1) : \mathcal{X} \to \mathcal{F}$ and $f(x_2) : \mathcal{X} \to \mathcal{F}$ from (1), minimise the worst-case output error bound*

$$||f_1(x_1) - f_2(x_2)||_2^2 \leq \gamma + \gamma_{x_1}\|x_1\|_2^2 + \gamma_{x_2}\|x_2\|_2^2 + \gamma_x\|x_1 - x_2\|_2^2, \quad \forall x_1, x_2 \in \mathcal{X}, \quad (5)$$

subject to $(\gamma_x,\, \gamma_{x_1},\, \gamma_{x_2}, \gamma) \in \mathbb{R}_+$.

The above problem establishes the general framework to address the specific problem of interest to this paper- that of bounding the worst-case error of a pruned or quantised neural network. To specialise Problem 1 to the computation of this NN quantisation bound, the second neural network in Problem 1 set to be the quantised one, as in

$$W_2^k = q(W_1^k), \quad b_2^k = q(b_1^k), \quad x_2 = q(x_1), \quad (6)$$

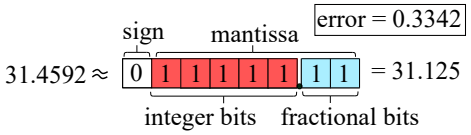
where $q(\cdot)$ is the quantisation function defined in Section 3.2. Likewise, upon considering pruned networks, Problem 1 is adapted by setting the second NN to be the pruned one, as in $W_2^k = p(W_1^k)$ where $p(\cdot)$ is a pruning operator, see Section 4.

3. Quantised neural networks

This section outlines a mathematical representation of quantisation when applied to neural network elements. With this representation, network quantisation can be incorporated into the neural network similarity problem of Problem 1, to bound the effect of additionally quantising the network input and neuron parameters.

3.1. Fixed-point arithmetic

It will be assumed that the quantised neural network's weights, biases and data are stored using fixed-point arithmetic, detailed in Yates (2009); Barrois and Sentieys (2017). The $\langle \texttt{IB}, \texttt{FB} \rangle$ format for fixed-point representations adopted in Gupta et al. (2015); Holi and Hwang (1993) is used, whereby $\texttt{IB}$ and $\texttt{FB}$ respectively denote the number of bits allocated for the integer and fractional parts of the number. As an example, the number 31.4592 is represented as



In this work, saturation of the representation is neglected. This is an important issue but could be avoided with the introduction of "temporary fixed-point registers with enough number of bits" (Gupta et al. (2015)).

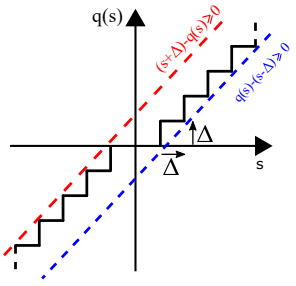
3.2. Quantisation function

The piece-wise constant quantisation function $q(\cdot)$ acting element-wise on vector and matrix arguments can be used to capture the fixed-point representation of the quantised neural network's weights, biases and data.

Given a real number $s \in \mathbb{R}$, the quantisation function is defined as

$$q(s) \triangleq \frac{\operatorname{sgn}(s)\lfloor |s| \cdot \Delta \rfloor}{\Delta}, \Delta = 2^{-\texttt{FB}}, \texttt{FB} \in \mathbb{Z}^+, \tag{7}$$

where $\lfloor \cdot \rfloor$ is the "floor" operator and Δ is the binary quantisation step, with the word length $\texttt{FB}$ setting the numerical precision of the quantisation. For the robustness analysis of the following sections, this function will be upper and lower bounded by



Decreasing $\texttt{FB}$ causes a coarse fixed-point representation, resulting in information being lost because of the quantisation and the neural network's output being corrupted, inducing an error.

Given a binary number (BN) represented in fixed-point format, the original decimal format (DN) can be recovered from

$$DN = \sum_{i=1}^{\mathtt{IB}-1} BN_{\langle I \rangle_i} * 2^i + \sum_{j=1}^{\mathtt{FB}-1} BN_{\langle F \rangle_j} * 2^{-j}, \tag{8}$$

with $\langle I \rangle$ and $\langle F \rangle$ denoting the integer and fractional parts of BN.

Note in this work the floor rounding scheme was used, i.e. $q(\cdot)$ maps an input value to the largest integer multiple of Δ smaller than the input value. Other rounding schemes exist and may be adapted into the proposed framework, e.g. round-to-nearest (Gupta et al. (2015)) and various piece-wise-linear and stochastic rounding methods.

As ReLU activation functions were adopted for the numerical results of this paper, the effect of additionally quantising the activation functions was ignored. However, this effect could be incorporated into the analysis as long as the quantised activation functions satisfy the quadratic constraints.

4. Pruned neural networks

Neural network pruning involves setting the weights or neurons considered to be the least important to zero, with the purpose of making the network structure sparse and so improve its computational and memory efficiency (Hooker et al. (2019)). A state-of-the-art pruning method is presented in Han et al. (2015) where a three-step framework is applied to train, prune and refine neural network parameters. Using this framework both AlexNet and VGG-16 have been successfully and effectively downsized with no loss in accuracy (Han et al. (2015)). In this paper, the operator $p(\cdot)$ denotes the pruning action implemented with a magnitude based approach, although other pruning approaches may also be implemented. By setting the second NN in (1) to be the pruned version, Problem 1 can be solved to compute the pruning error bounds.

5. Abstracting the nonlinearities and sets using quadratic constraints

In general, solving Problem 1 is intractable because of: (i) the potentially non-convex mapping of each NN with nonlinear activation functions, and (ii) the bounds are required to hold robustly for all inputs in the pre-defined set $(x_1, x_2) \in \mathcal{X}$ with $\mathcal{X}$ allowed to be infinite dimensional. Using quadratic constraints, this paper is able to provide an upper bound for the error from neural network pruning and quantisation by producing an algebraic, convex outer approximation of the various sets and nonlinearities of the problem. This allows solutions to Problem 1 to be computed via a SDP, with the conservatism of the approach coming from the fact that it only implicitly characterises the various nonlinearities and sets of the analysis. The remaining part of this section details the quadratic constraints for the input set, the nonlinear activation functions, the quantisation and the error bound (14).

5.1. Abstraction of the input set

The various quadratic constraints satisfied when the NN inputs are contained to the set $(x_1, x_2) \in \mathcal{X}$ are detailed in (Li et al., 2021, Appendix 3). When the inputs are constrained to this set, there exists a matrix $P_{\mathcal{X}} \in \mathbb{R}^{2(n_x+M)+1 \times 2(n_x+M)+1}$ built from the matrix variables (the lambdas of (Li et al.,

2021, Definition 3)) such that

$$M_{\mathcal{X}}(P_{\mathcal{X}}) = \eta^T P_{\mathcal{X}} \eta \geq 0, \quad \forall (x_1,\, x_2) \in \mathcal{X}. \tag{9}$$

5.2. Abstraction of the activation functions

Dealing with the nonlinear activation functions is the main source of difficulty in generating robust solutions to Problem 1. Thankfully, most of the commonly used NN activation functions (e.g. those mentioned in (Li et al., 2021, Table 1)) have properties that allow them to be characterised by quadratic constraints. The mathematical definitions of these properties can be found in (Li et al., 2021, Definition 4) and the associated quadratic constraints are given in (Li et al., 2021, Lemma 5).

Using the compact NN representation of (3), the various quadratic constraints satisfied by the activation functions of (Li et al., 2021, Lemma 5) can be collected together and represented as a single inequality

$$M_{\phi}(\Lambda) = \mu^T \Lambda \mu = \eta^T \left(E^T \Lambda E\right) \eta \geq 0, \quad \forall (x_1,\, x_2) \in \mathcal{X}. \tag{10}$$

Here, the matrix

$$\Lambda = \begin{bmatrix} \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} & \Lambda_{13} & \Lambda_{14} & \Lambda_{15} \\ \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} & \Lambda_{23} & \Lambda_{24} & \Lambda_{25} \\ \Lambda_{13}{}^T & \Lambda_{23}{}^T & \Lambda_{33} & \Lambda_{34} & \Lambda_{35} \\ \Lambda_{14}{}^T & \Lambda_{24}{}^T & \Lambda_{34}{}^T & \Lambda_{44} & \Lambda_{45} \\ \Lambda_{15}{}^T & \Lambda_{25}{}^T & \Lambda_{35}{}^T & \Lambda_{45}{}^T & \Lambda_{55} \end{bmatrix} \tag{11}$$

is composed of elements $\Lambda_{i,j}, \{i,j\} \in \{1,2,3,4,5\}$ being linear combinations of the scaling variables of the various quadratic constraints of $\phi(\cdot)$, e.g. the lambdas of (Li et al., 2021, Lemma 5).

Note that information on the quantisation and pruning of the NN parameters (weights and biases), or any other alteration in value or structure to the parameters in general, is introduced a priori into the SDP problems' constraints via the abstractions just defined, and therefore does not require extra QCs.

5.3. Abstraction of the quantisation function

Besides the activation functions $\phi(\cdot)$, for quantised NNs the quantisation function $q(x_i)$ from (7) acting on the input data is another nonlinearity that will have to to be accounted for in the analysis if error bounds are to be obtained. Thankfully, this nonlinear function also satisfies certain quadratic constraints as defined in (Li et al., 2021, Lemma 7). These quadratic constraints can be combined into the single inequality

$$M_q(P_q) = \eta^T P_q\, \eta \geq 0, \quad \forall (x_1,\, x_2) \in \mathcal{X}, \tag{12}$$

for some matrix variable P_q built from the various lambdas of (Li et al., 2021, Lemma 7).

5.4. Abstraction of the error bound

The bound for the worst-case error between the two neural networks

$$||f_1(x_1) - f_2(x_2)||_2^2 - (\gamma + \gamma_{x_1}\|x_1\|_2^2 + \gamma_{x_2}\|x_2\|_2^2 + \gamma_x\|x_1 - x_2\|_2^2) \leq 0, \quad \forall (x_1, x_2) \in \mathcal{X} \tag{13}$$

can also be expressed as a quadratic

$$M_{\|f-g\|_2}(\Gamma) = \eta^T V^T \Gamma(\gamma_{x_1}\, \gamma_{x_2},\, \gamma_x,\, \gamma)\, V\eta \leq 0, \quad \forall (x_1,\, x_2) \in \mathcal{X}, \tag{14}$$

where

$$\Gamma = -\begin{bmatrix} (\gamma_{x_1}+\gamma_x)\mathbb{I}_{n_{x_1}} & -\gamma_x \mathbb{I}_{n_{x_1}} & \mathbf{0}_{n_{x_1}\times n_{f_1}} & \mathbf{0}_{n_{x_1}\times n_{f_2}} & \mathbf{0}_{n_{x_1}} \\ -\gamma_x \mathbb{I}_{n_{x_2}} & (\gamma_{x_2}+\gamma_x)\mathbb{I}_{n_{x_2}} & \mathbf{0}_{n_{x_2}\times n_{f_1}} & \mathbf{0}_{n_{x_2}\times n_{f_2}} & \mathbf{0}_{n_{x_2}} \\ \mathbf{0}_{n_{f_1}\times n_{x_1}} & \mathbf{0}_{n_{f_1}\times n_{x_2}} & \mathbb{I}_{n_{f_1}} & -\mathbb{I}_{n_{f_1}} & \mathbf{0}_{n_{f_1}} \\ \mathbf{0}_{n_{f_2}\times n_{x_1}} & \mathbf{0}_{n_{f_2}\times n_{x_2}} & -\mathbb{I}_{n_{f_2}} & \mathbb{I}_{n_{f_2}} & \mathbf{0}_{n_{f_2}} \\ \mathbf{0}_{n_{x_1}}{}^T & \mathbf{0}_{n_{x_2}}{}^T & \mathbf{0}_{n_{f_1}}{}^T & \mathbf{0}_{1\times n_{f_2}}{}^T & \gamma \end{bmatrix}, \tag{15}$$

and

$$V = \begin{bmatrix} \mathbb{I}_{n_{x_1}} & \mathbf{0}_{n_{x_1}\times n_{x_2}} & \mathbf{0}_{n_{x_1}\times M} & \mathbf{0}_{n_{x_1}\times M} & \mathbf{0}_{n_{x_1}} \\ \mathbf{0}_{n_{x_2}\times n_{x_1}} & \mathbb{I}_{n_{x_2}} & \mathbf{0}_{n_{x_2}\times M} & \mathbf{0}_{n_{x_2}\times M} & \mathbf{0}_{n_{x_2}} \\ \mathbf{0}_{n_{f_1}\times n_{x_1}} & \mathbf{0}_{n_{f_1}\times n_{x_2}} & \begin{bmatrix}\mathbf{0}_{n_{f_1}\times M-n_{\ell_1}} & W_1^\ell\end{bmatrix} & \mathbf{0}_{n_{f_1}\times M} & b_1^\ell \\ \mathbf{0}_{n_{f_2}\times n_{x_1}} & \mathbf{0}_{n_{f_2}\times n_{x_2}} & \mathbf{0}_{n_{f_2}\times M} & \begin{bmatrix}\mathbf{0}_{n_{f_2}\times M-n_{\ell_2}} & W_2^\ell\end{bmatrix} & b_2^\ell \\ \mathbf{0}_{n_{x_1}}{}^T & \mathbf{0}_{n_{x_2}}{}^T & \mathbf{0}_M{}^T & \mathbf{0}_M{}^T & 1 \end{bmatrix}. \tag{16}$$

Here, for a fair comparison, implies $n_{x_1} = n_{x_2}$ and $n_{f_1} = n_{f_2}$, as in both NNs have the same number of inputs and outputs.

6. Neural network error bounds as a semidefinite programme

With the QCs defined for each feature of the problem, upper bounds to (5) in Problem 1 can be computed from the SDP of Theorem 1. Worst-case error bounds for pruned neural networks can be obtained directly upon setting the second network of the problem to be the pruned one. The specialisation to quantised neural networks has the quantisation quadratic constraints of (12) included within the linear matrix inequality (17b).

Theorem 1 *Consider the two neural networks defined in* (1) *satisfying the quadratic constraints of* (9), (10) *and* (12). *Set the weights* $(w_{x_1},\, w_{x_2},\, w_x,\, w_{\mathit{aff}}) \in \mathbb{R}_+$ *scaling the relative importance of the quadratic and affine terms in the error bound* (13). *If there exists a solution to*

$$\begin{aligned} \textit{min} \quad & w_{x_1}\gamma_{x_1} + w_{x_2}\gamma_{x_2} + w_x\gamma_x + w_{\mathit{affine}}\gamma, && \text{(17a)} \\ \textit{subject to} \quad & P_{\mathcal{X}}(\cdot) + P_q(\cdot) + E^T\Lambda(\cdot)E + \Gamma(\cdot) \prec 0, && \text{(17b)} \\ & \gamma_{x_1} \geqslant 0,\ \gamma_{x_2} \geqslant 0,\ \gamma_x \geqslant 0,\ \gamma \geqslant 0. && \text{(17c)} \end{aligned}$$

with the matrix variables $P_{\mathcal{X}}(\cdot)$, $P_q(\cdot)$, $\Lambda(\cdot)$ *and* $\Gamma(\cdot)$ *defined in* (9), (12), (10) *and* (14), *then the error is bounded from above by*

$$\|f_1(x_1) - f_2(x_2)\|_2^2 \leq \gamma + \gamma_{x_1}\|x_1\|_2^2 + \gamma_{x_2}\|x_2\|_2^2 + \gamma_x\|x_1 - x_2\|_2^2, \quad (x_1,\, x_2) \in \mathcal{X}. \tag{18}$$

Proof See (Li et al., 2021, Appendix 6). ■

7. Numerical examples and discussion

In this section, the effectiveness of the obtained bounds was explored with several numerical examples. The first subsection presents examples for bounding the error between two generic neural networks, and the second is focused on quantised and pruned networks.

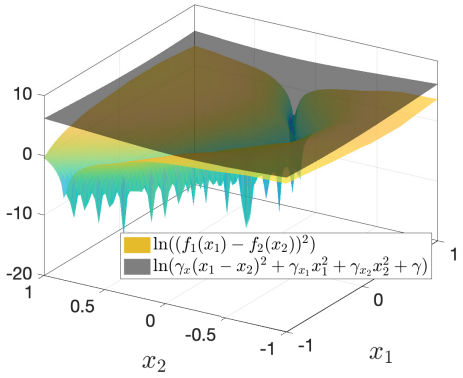
In all the examples, the input was constrained to the hyper-cube $(x_1, x_2) \in \mathcal{X} = \{x_1, x_2 \in \mathbb{R}^{n_x} \mid |x_1|, |x_2| \leq \overline{x}\}$, where $\overline{x} = 1$ and the ReLU was chosen as the activation function. For all examples, the SDP of Theorem 1 was solved with the MOSEK (ApS, 2019) solver implemented through the YALMIP (Löfberg, 2004) interface of MATLAB. The code and the various models used in the examples can be found at: https://github.com/ElrondL/Robust-Quantisation-Bounds.

7.1. Neural network similarity bounds

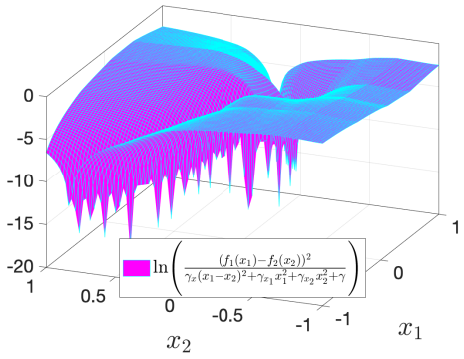
For the first experiment, the performance of the framework defined in Theorem 1 was checked by comparing randomly generated neural networks with $n_x = 1$ as the input dimension, $n_f = 1$ as the output dimension and $\ell = 2$ hidden layers, each with $n_k = 10$ neurons. The two neural networks were fed with randomly generated signals $\{x_1, x_2\} \in \mathcal{X}$. For each NN layer dimension (ℓ), 100 random neural networks were generated and the average γ_{x_1}, γ_{x_2}, γ_x, γ, mean bound tightness, maximum bound tightness, minimum bound tightness, and runtime were recorded. Tightness of the bound was defined as the natural log of the error between the output difference norm and the over approximation bound

$$T = \ln((f(x_1) - f(x_2))^2) - \ln(\gamma_{x_1} x_1^2 + \gamma_{x_2} x_2^2 + \gamma_x (x_1 - x_2)^2 + \gamma), \tag{19}$$

with the results given in (Li et al., 2021, Table 2).



(*a*) The error (coloured) and bound (grey) surfaces.



(*b*) The tightness surface T of the bound and error surfaces in Fig.1(a)

Figure 1: Example neural network error bound with $\ell = 2$.

A bounding surface holding robustly for all inputs generated was obtained for each test case, an example of which can be found in Figure 1. When $n_x = 1$ and $n_f = 1$, the bound follows the error surface with $T \leq 2$ for the majority of the input space. The singularity of the log function at zero means that points in the input space generating small errors between the two neural networks are associated with the deep valleys in the figure. The average distance between the bound and the error surface became larger with an increasing number of neurons and for multi-dimensional input and output vectors, as reflected in the data of (Li et al., 2021, Table 2).

7.2. Quantisation error bound

Error bounds were also computed for quantised NNs where $W_2 = q(W_1)$, $b_2 = q(b_1)$ and $x_2 = q(x_1)$. Here, the output dimension was $n_f = 1$, the input dimension was $n_x = 1$ with the input set for x_1 consisting of 100 evenly spaced values in $[-1, 1]$. The quantisation step was $\Delta = 2^{-2}$, as in two bits were used to store each number. (Li et al., 2021, Figure 3) contains example error bound curves for the various models considered and also an example in 3D (where the condition $x_2 = q(x_1)$ was relaxed, giving a looser bound as the input space was less restricted).

Bounds were then computed for 100 randomly generated NNs with $\ell = 4$ hidden layers, with the corresponding results shown in (Li et al., 2021, Table 3). Bound tightness was defined as the error between the original and quantised networks' output difference norm and the over-approximation bound of (19).

A bounded curve holding for all inputs was generated for each test case, with the step patterns of the error curves following from the quantisation. It was also observed that the bound loosened with an increasing number of neurons, as seen in (Li et al., 2021, Table 3).

Worst-case quantisation error bound

The worst-case quantisation error and bound values were also recorded and averaged for 100 random neural networks of $\ell = 2$ layers for quantisation levels $\Delta = 2^i, i \in \{1, 2, 3, 4, 5\}$. Note that the bound and error values come in pairs, so there exists a bound value corresponding with the maximum error as well as an error value corresponding to the maximum bound value. These results where then plotted in (Li et al., 2021, Figure 4) where the maximum bound values tended to be close to the observed maximum error, however the gap between the corresponding maximum error and the bound tended to be quite large. It was also observed that both the error and the bound decreased at similar rates with an increasing level of quantisation.

Pruned neural networks

Figure 5 in (Li et al., 2021) shows the results for bounding the error between a randomly-generated 20 neuron neural network and its pruned version (with $f_2(x_2)$ in Problem 1 being the pruned one), with the weights of the eight hidden neurons with smallest 2-norms set to zero (Blalock et al. (2020)). Note, that by setting the second neural network to be identical to the first, the framework could also bound the behaviour of a neural network within its input space. Pruning that removes neurons rather than setting weights to zero could also be bounded with this framework.

Conclusions

A method to generate uppers bounds for the worst-case error induced by either quantising or pruning a trained neural network was introduced. The bounds hold robustly for all inputs in pre-specified sets, account for nonlinear activation functions and are generated from the solution of a semi-definite programme. Several numerical examples graphically illustrated the tightness of the bounds. Future work will explore reducing the conservatism of the bounds, improving the scalability of the semi-definite programming solvers and applying the method to other problems of interest where the similarity of two neural networks is to be evaluated.

Acknowledgments

Ross Drummond would like to thank the Royal Academy of Engineering for funding this research through a UKIC Fellowship as well as the Nextrode project of the Faraday Institution. Jiaqi Li would like to thank the Department of Engineering, University of Oxford for funding through the UROP scheme.

References

MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. URL http://docs.mosek.com/9.0/toolbox/index.html.

B. Barrois and O. Sentieys. Customizing fixed-point and floating-point arithmetic — A case study in K-means clustering. In *International Workshop on Signal Processing Systems (SiPS)*, pages 1–6, 2017. doi: 10.1109/SiPS.2017.8109980.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33, 2020.

Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: the confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 2020.

Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *arXiv preprint arXiv:1903.01287*, 2019.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Deniz Gündüz, Paul de Kerret, Nicholas D Sidiropoulos, David Gesbert, Chandra R Murthy, and Mihaela van der Schaar. Machine learning in the air. *IEEE Journal on Selected Areas in Communications*, 37(10):2184–2199, 2019.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746. PMLR, 2015.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

J. L. Holi and J. . Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3):281–290, 1993. doi: 10.1109/12.210171.

Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248*, 2019.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016.

Jiaqi Li, Ross Drummond, and Stephen R Duncan. Robust error bounds for quantised and pruned neural networks. *arXiv preprint arXiv:2012.00138*, 2021.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

J. Löfberg. Yalmip : A toolbox for modelling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B Letaief. Communication-efficient edge AI: Algorithms and systems. *arXiv preprint arXiv:2002.09668*, 2020.

Sungho Shin, Kyuyeon Hwang, and Wonyong Sung. Fixed-point performance analysis of recurrent neural networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 976–980. IEEE, 2016.

Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization. *arXiv preprint arXiv:1511.06488*, 2015.

Randy Yates. Fixed-point arithmetic: An introduction. *Digital Signal Labs*, 81(83):198, 2009.

George Zames. On the input-output stability of time-varying nonlinear feedback systems part one: Conditions derived using concepts of loop gain, conicity, and positivity. *IEEE Transactions on Automatic Control*, 11(2):228–238, 1966.

Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287, 2019.

Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8): 1738–1762, 2019.