Summer 2022

On Incorporating the Stochasticity of Quantum Machine Learning Into Classical Models

Joseph Lindsay

On Incorporating the Stochasticity of Quantum Machine Learning into Classical Models

by

Joseph Lindsay

Bachelor of Science
University of South Carolina 2020

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in

Computer Science

College of Engineering and Computing

University of South Carolina

2022

Accepted by:

Ramtin Zand, Director of Thesis

Stephen Fenner, Reader

Forest Agostinelli, Reader

Tracey L. Weldon, Interim Vice Provost and Dean for the Graduate Education

Abstract

While many of the most exciting quantum computing algorithms are currently impossible to be implemented until fault-tolerant quantum error correction is achieved, noisy intermediate-scale quantum (NISQ) devices allow for smaller scale applications that leverage the paradigm for speed-ups to be researched and realized. A currently popular application for these devices is quantum machine learning (QML). Recent works over the past few years indicate that QML algorithms can function just as well as their classical counterparts, and even outperform them in some cases. Many current QML models take advantage of variational quantum algorithm (VQA) circuits, given that their scale is typically small enough to be compatible with NISQ devices and the method of automatic differentiation for optimizing circuit parameters is familiar to machine learning. As with many skeptics on its benefits of quantum computing in general, there is some concern as to whether machine learning is the "best" use case for the advantages that NISQ devices make possible. To this end, the nature of this work is to investigate the utilization of stochastic methods inspired by QML in attempt to approach the reported successes in performance. Using the long short-term memory (LSTM) model as a case study and by analyzing the performance of classical, stochastic, and QML methods, this work aims to elucidate if it is possible to achieve QML's benefits on classical machines by incorporating aspects of its stochasticity.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Through initial questioning by Feynman [23] in eighties, it has become a known fact that it is a computationally expensive task to simulate quantum systems. Following this discovery, we have since learned that there exists classifications of problems that can be solved in polynomial time with computation based on quantum mechanics that have no known polynomial time solution on classical, or von Neumann, machines [30, 63, 24, 62]. This comes from a large corpus of works throughout the eighties and nineties, encompassing research done by Bernstein and Vazirani [9], Simon [55], Shor [54], Yao [62], and many others [5, 6, 27]. The best known example of such problems is that of prime integer factorization, the solution for which was shown by Peter Shor in 1994. He formulated a quantum algorithm to compute the order r of an element $x \pmod N$ in only $\mathcal{O}(\lceil lg(N) \rceil^3)$ operations, which can then be used to find the prime factors p and q of N by $gcd(x^{r/2} \pm 1, N)$ [54, 57, 42]. This is a massive increase in efficiency over any known classical algorithm that serves the same purpose, e.g. the "number field sieve" which performs the factoring in $\mathcal{O}(e^{((64/9)^{1/3}+\mathcal{O}(1))(lg(N))^{1/3}(lg(lg(N)))^{2/3}})$ operations [8, 13]. Unfortunately, constructing fault-tolerant quantum systems capable of running the most promising algorithms at scales that matter is currently infeasible and likely still many years away. This is due to the technological difficulties in the implementation of hardware systems, which primarily consist of the scalability and decoherence times of physical qubits used [44].

Despite this, some quantum algorithms are simple enough to be handled relatively

well on modern noisy intermediate-scale quantum (NISQ) devices. Here, "noisy" refers to the lack of fault-tolerant quantum error-correction and "intermediate-scale" indicates that the aforementioned scalability and coherence issues are present. This allows researchers to leverage the paradigm even at a lower level and progress the state-of-the-art in computational efficiency, and can also be described as near-term quantum computation [10]. The majority of these devices exist as part of hybrid quantum-classical systems, and there are a variety of methods to take advantage of such systems. An area of current near-term quantum computing research that has experienced popularity in recent years, and is relevant to this work, is the use of such systems for machine learning.

Aside from the novelty of simply applying quantum computing to machine learning tasks, quantum machine learning (QML) overall has shown promise in providing a quantum advantage to several learning subroutines and models [11, 21, 48, 56, 20, 51, 35, 52, 3, 4]. Although there exists various formulations to implement QML methods, many are being realized through the use of variational quantum algorithms (VQA). These VQAs function similar to automatic differentiation techniques, such as what is commonly applied in neural network learning. Parametrized quantum gates are variationally updated and classically optimized such that the final measurement upon the circuit delivers the desired result [41, 53, 39]. These algorithms enable a quantum advantage for several problems, including simulation of dynamical quantum systems, combinatorial optimization, and chemistry problems; however, the likeness to machine learning methods makes their application to such systems natural [40, 16].

As an example, Zoufal, Lucchi, and Woerner [68] demonstrates a variational quantum boltzmann machine (VarQBM) that produces results and performance metrics competitive to all of the classic classifiers compared against this model for the chosen application, with the VarQBM even having the best F_1 score by a margin of 0.04.

Another such example has been shown by Chen, Yoo, and Fang [17], wherein they give the construction and theory of a variational quantum long short-term memory (QLSTM) model and demonstrate through several experiments that it is capable of converging faster and generally achieving lower loss values than its classical version.

These are exciting results for the prospects of QML; even with an inherently stochastic system, these models are able to achieve results that demonstrate better performance and convergence rates than classical methods in many cases. This brings to mind an important question: **Can we achieve similar results to variational QML by incorporating stochasticity inspired by its architectures into classical machine learning models?** If this were the case, then perhaps some of these advantages could be achieved without the need for quantum hardware. Without invalidating the area of study, this could provide an indication on whether QML is truly an important application for near-term quantum computing research and utilization. Following this introduction, Chapter 2 establishes the necessary foundation for the case study employed to investigate this matter; Chapter 3 details the techniques used to explore this question; Chapter 4 provides brief implementation details and analyzes the results achieved; Chapter 5 discusses the implications of this work's findings; and finally Chapter 6 concludes the study with the author's final thoughts and potential future work on the matter.

Chapter 2

Background of Case Study

To more closely examine the impact that stochasticity has on the reported advantages QML exhibits over classical machine learning (ML), it is beneficial to assess the performance of a particular model through a case study. With the backing of existing research and literature as previously described in Chapter 1, the long short-term memory (LSTM) model was an attractive option and thus selected for this purpose [17]. Given this, Section 2.1 reviews the construction and operation of a LSTM model, followed by Sections 2.2 and 2.3 which, respectively, explain the metrics used to gauge the model's performance and the optimization methods employed. Then in Section 2.4, the experiments that were selected for the study are presented. Lastly, Section 2.5 provides a brief introduction to quantum computing and concepts that will be relevant to the work.

2.1 Long Short-Term Memory Model

As a variant of the recurrent neural network (RNN), the LSTM model is best suited for analyzing time-series data sequences to learn temporal dependencies. Like most ML methods, data is passed into the model and undergoes a series of multiply-and-accumulation (MAC) operations followed by certain activation functions. RNN models like the LSTM are differentiated from other types of neural networks by passing an additional parameter, referred to as the hidden state, into each iteration of the model's recurrent portion, referred to as a cell. The hidden state is derived by the model's cell from the previous time step data in the time-series. Much like the weights of typical neural networks, the hidden state has a dimension or size; this parameter initially starts as a series of zeroes with the specified dimension. This hidden state is usually taken as the cell's output as well, potentially followed by some post-processing steps within the model. While this may simply be an activation function, its not uncommon to send the output through a fully-connected layer or a larger network architecture.

While RNNs and their variants can be depicted as looping a single cell's hidden state onto itself with data input for the each time step, the model can be logically "unrolled" into many cells, the specific number of which is determined by the batch size of the network. In cases where only a single prediction is made per batch, this batch size can be interpreted as the number of time steps analyzed before a prediction is made. It should be noted that all cells when "unrolled" utilize the same set of parameters, giving rise to the "looping" representation of the network and its usefulness in analyzing time-series data. These representations for a standard RNN are shown by Figure 2.1.

The primary differences that make LSTM networks distinct from standard RNNs are the equations used within the network's cell and the addition of another parameter, referred to as the cell state, passed to subsequent time steps of the network's

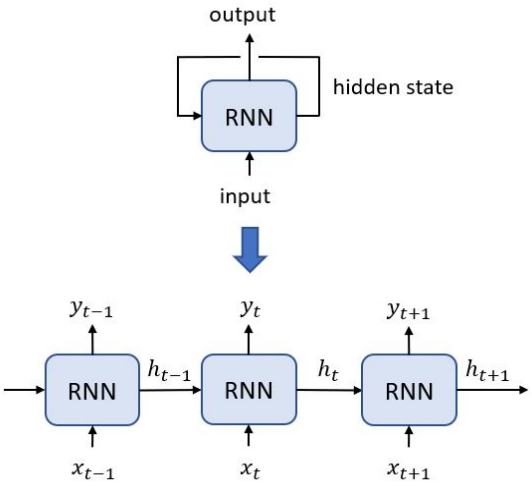


Figure 2.1 Graphical representations of a standard RNN model.

current batch like its hidden state. These changes effectively address issues that standard RNNs have with vanishing gradients and allows the model to learn longer sequential dependencies in the data. A classic LSTM cell can be mathematically expressed by the following equations:

$$f_t = \sigma(W_f \cdot v_t + b_f) \tag{2.1a}$$

$$i_t = \sigma(W_i \cdot v_t + b_i) \tag{2.1b}$$

$$\tilde{C}_t = tanh(W_{\tilde{C}} \cdot v_t + b_{\tilde{C}}) \tag{2.1c}$$

$$c_t = (f_t * c_{t-1}) + (i_t * \tilde{C}_t) \tag{2.1d}$$

$$o_t = \sigma(W_o \cdot v_t + b_o) \tag{2.1e}$$

$$h_t = o_t * tanh(c_t) \tag{2.1f}$$

Where v_t is the previous hidden state concatenated with the current data s.t. $v_t = [h_{t-1}, x_t]$, the function of an LSTM cell can be described by the three gates it consists of: the forget gate, the input and update gate, and the output gate.

1. Corresponding to Equation 2.1a and the first part of Equation 2.1d, the forget gate comes first and works to update the relevance of data from the previous cell state depending upon the current time step data. The concatenated data v_t undergoes a MAC operation with weights W_f and biases b_f before receiving a sigmoid activation. The result f_t is then multiplied element-wise with the previous cell state c_{t-1}.

2. Comprised by Equations 2.1b, 2.1c, and the latter half of 2.1d, the input and update gate is the next portion of the LSTM cell and functions to decide what new information the current cell state should be updated with. The concatenated data v_t receives two separate MAC operations: one with weights W_i and biases b_i, and the other with weights $W_{\tilde{C}}$ and biases $b_{\tilde{C}}$. The former determines which data values of the cell state should be updated and gets a sigmoid activation, while the latter provides an initial candidate with which to update the cell state and gets a hyperbolic tangent activation. The resulting values i_t and $\tilde{C}_t$ are then multiplied together element-wise to produce the scaled candidate values, before being added element-wise to the cell state resulting from the forget gate.

3. Consisting of Equations 2.1e and 2.1f, the final portion of the LSTM cell, the output gate, sets the current hidden state of the network before passing it along with the cell state to the next time step, and potentially returning that hidden state as output. Once more, the concatenated data v_t receives a MAC operation, this time with weights W_o and biases b_o, then gets a sigmoid activation to decide what values to output. The current cell state c_t then receives a hyperbolic tangent activation and is multiplied element-wise with the result o_t of the previous operation. This produces the current hidden state h_t for the time step; as mentioned, this and c_t will become h_{t-1} and c_{t-1} for the next time step

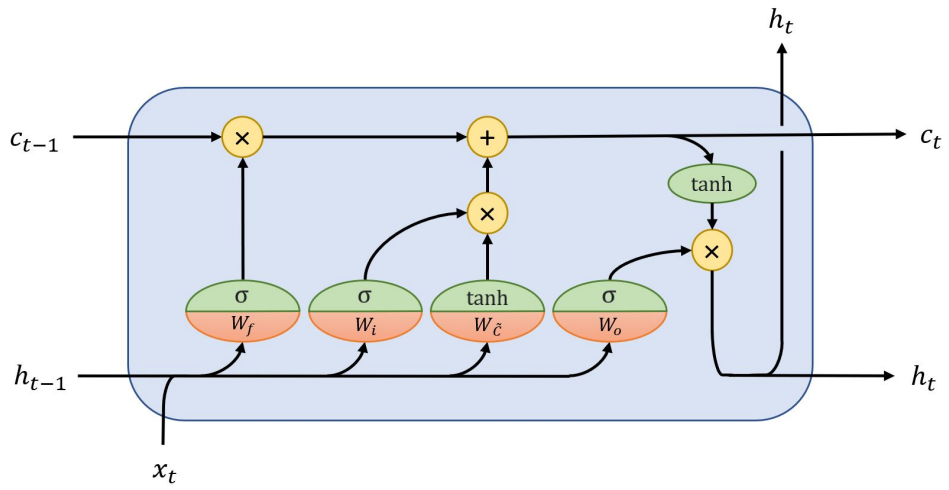


Figure 2.2 Graphical representation of an LSTM cell.

and may also be returned as output y_t.

This construction of an LSTM cell as described above is illustrated by Figure 2.2. It is worth mentioning that each of the MAC operations occur with separate sets of weights and biases, indicated by their subscripts [25, 33, 17, 45].

2.2 Metrics

As with any ML model, it is necessary to evaluate the model's performance using an established set of metrics. This allows us to gauge how effectively the model has learned the data that it has been trained upon. Generally, the chosen metrics can be determined by whether the intended goal is classification or regression; while it is possible to configure an LSTM model to perform classification, they are more commonly used for regression based tasks and have been utilized as such in this work. The study herein has been assessed using three well-known metrics: mean square error (MSE), root mean square error (RMSE), and coefficient of determination (R^2 score).

One of the most widely-used metrics for regression, MSE provides a measure as to how far the residual errors of the model extend from the ideal fit of the data.

This weighs large errors heavier than small ones, making it suitable for finely honing parameters during optimization. As such, MSE has been chosen as the loss metric throughout this work. Where $\hat{y}_i$ is the model prediction, y_i is the ground truth for the corresponding sample, and n is the total number of labels, this metric can be expressed by the following equation:

$$\text{MSE} = \sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n} \tag{2.2}$$

As its name implies, RMSE is very closely related to MSE and essentially expresses the same information regarding the model. While MSE's square units penalize larger errors more heavily however, RMSE's square root returns the metric's units to that of the ground truth data which enables easier analysis. This metric can be mathematically found using the following equation:

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.3}$$

While the previous two metrics are useful for understanding the standard deviation, the R^2 score, or more simply just R^2, gives an indication to the variance from ground truth present within the results. Essentially, this metric provides a rating for how well a model is able to account for the variance present within the data it is shown. The values for this are typically in the range [0.0, 1.0), but particularly bad fits for the regression may have a negative value. Normally, a higher R^2 is a good thing, but a lower value doesn't necessarily mean that the model is bad. Where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$, the coefficient of determination can be calculated with the following equation:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.4}$$

2.3 Optimization

In order to perform learning tasks with neural networks, there must be way to update the network parameters in a manner that minimizes the error between actual output and expected target values. The standard method used by the ML community to accomplish this, since it was introduced by Rumelhart, Hinton, and Williams [50] in 1986, is a process known as back-propagation or simply backprop. The general idea of this method is to iteratively perform a gradient descent in the task's loss landscape until the total error has been optimally minimized. After a forward pass through a given model, the total error is calculated using a loss function like MSE as described in Section 2.2. The update value for parameters is then obtained by computing the partial derivative of the total error with respect to the corresponding parameter. This is accomplished by first finding the partial derivative of the error with respect to the network's output and then repeatedly applying the chain rule of differentiation until the desired partial derivative is reached. Propagating the error gradient backwards through the model's equations in such a manner is why the term "back-propagation" is used to describe this procedure.

For this method to be effective for LSTM models, it needs to be adapted into a variant suited for temporal learning tasks over time-series data. The common approach used is known as back-propagation-through-time (BPTT), the basis of which is fairly simple once standard backprop is understood. The general procedure is to "unroll" the network as described in Section 2.1 and propagate the error gradient through every time step, accumulating the parameter updates until the full sequence has been processed. While BPTT is formally applied in learning tasks where the entire dataset is trained at once, it can be applied to batched datasets as truncated BPTT (TBPTT). TBPTT is delineated from BPTT by allowing a different number of time steps for analysis and error accumulation; however, most forms of TBPTT are out of the scope for this work and it can be generally considered as a version of

BPTT applicable to batches within full training epochs [60, 61].

Once the error gradients have been accumulated, regardless of the training method, an optimization routine is then used to actually apply the parameter updates. *Standard gradient descent* can be applied using the equation:

$$\Delta p = -\alpha \frac{\delta E}{\delta p}, \tag{2.5}$$

where p refers to the parameter for update, E is the total error, and α is some small learning rate. While this scheme is generally effective, it can be prone to converging on a local rather than global minimum. Furthermore, convergence is dictated by the learning rate; high values could cause parameters to oscillate around minima, and low values will result in slower convergence times and a greater potential of becoming trapped in a local minima. Thankfully, these difficulties can be handled by utilizing a more sophisticated optimization algorithm.

There are a variety of options available for this purpose, but this work takes the cue of Chen, Yoo, and Fang [17] and uses the root mean square propagation (*RMSProp*) method. Proposed by Hinton and Tieleman [29] in 2012 during a Coursera course and famous for not being published, *RMSProp* implements an adaptive learning rate by keeping a moving average of the squared gradients and dividing the learning rate by the square root of the result. This is implemented using the following equations:

$$M_t = \rho \cdot M_{t-1} + (1 - \rho) \left(\frac{\delta E}{\delta p} \right)^2 \tag{2.6a}$$

$$\Delta p = -\frac{\alpha}{\sqrt{M_t + \epsilon}} \frac{\delta E}{\delta p} \tag{2.6b}$$

where M is the moving average, ρ is the decay parameter for M, and ϵ is a small smoothing term that aims to avoid division by 0.

2.4 Experiments

As mentioned in Section 2.1, LSTM models are innately suited for analyzing temporal dependencies and patterns in sequential data. This has applications in areas such as

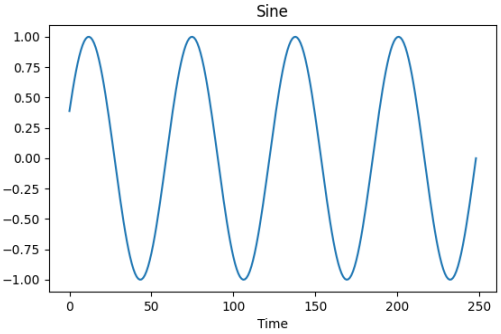


Figure 2.3 *Sine* function over approximately 4 periods.

natural language processing, trend analysis, speech recognition, anomaly detection, and much more. For the purposes of this work, however, several simple experiments have been chosen in order benchmark model performance. All experiments presented in this work can be categorized as trials in pattern recognition. Elaborated on individually below, these can be broadly classified as either periodic or inspired by physical dynamics.

A common practice when learning and building LSTM models is to test them using simple periodic functions, an approach which was adopted herein. Within the family of trigonometric functions, *sine* is perhaps the simplest, and as such is commonly used to confirm the validity of implementations. Therefore, all models have been initially verified using trigonometric *sine* function, producing a basic sinusoidal wave as shown in Figure 2.3 for the models to learn.

To introduce a further level of intricacy, a non-smooth periodic function was also tested. With the additional benefit of also being non-continuous, a *sawtooth* waveform as shown in Figure 2.4 was selected. While a simple pattern, disrupting the direct linearity exhibited by the function between periods is a complexity that poses an interesting challenge. Features such as this have been found to be difficult for ML models to recognize.

These two functions are suitable for benchmarking sequential data ML models,

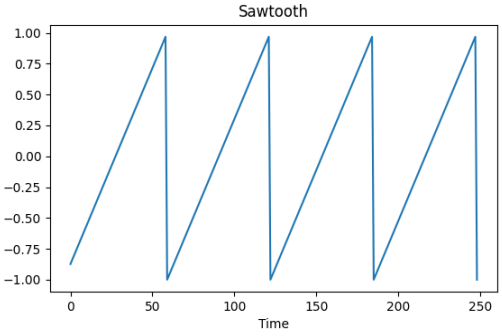


Figure 2.4 *Sawtooth* function over 4 periods.

but are rather simple within the realm of periodic functions. As such, the next experiment desired to be chosen for this work was a periodic system that has increased the complexity compared to two previously introduced. The function resulting from *summed cosine waves* was suitable for this purpose, incorporating an additional layer of mathematics onto the standard trigonometric function. To greater increase this supplementary complexity, the data points passed to the cosine functions receive a multiplicative factor to alter the waves' frequency. This general formulation can be extended to the sum of many cosine waves, but herein only two are utilized and thus can be expressed mathematically by the equation

$$y(x) = A_1 cos\left(\frac{2\pi x}{\lambda_1}\right) + A_2 cos\left(\frac{2\pi x}{\lambda_2}\right). \tag{2.7}$$

Here A_1 and A_2 refer to the two waves' amplitudes, while λ_1 and λ_2 are the waves' wavelengths. The parameters used in the configuration for this work are $A_1 = A_2 = 1$, $\lambda_1 = 9$, and $\lambda_2 = 11$. The specific waveform used for this study as indicated by Equation 2.7 and the given parameters can be visualized by Figure 2.5. Despite its periodic nature, this method provides an important basis for a classic physics system known as wave packets. In the case of sound waves, the concept of infinitely repeating waves of the form described is referred to as beats; this becomes a wave packet when localized to a single pulse, which is a core foundation for fields such as audio signal processing and recognition. Consequently, this experiment can be regarded as both

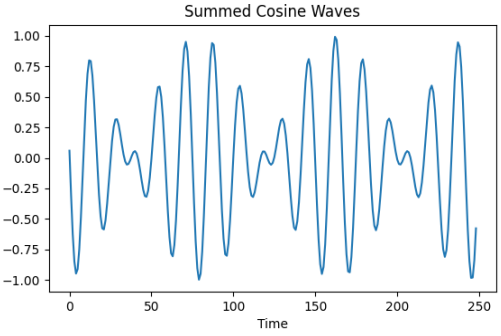


Figure 2.5 *Summed cosines waves* function.

a periodic function and a simple physics-based system.

Although these functions provide simple patterns to train and evaluate models on, all are periodic and repeat. As such, a non-periodic function was desired to ensure the models are robust in ability. Following the previous example introduced, a system of physical dynamics was an attractive option for this purpose. Once more taking a cue from Chen, Yoo, and Fang [17], the *damped harmonic oscillator* physics system was a suitable candidate for the final experiment selected. The motion of such a system can be defined by the differential equation

$$\frac{d^2x}{dt^2} + 2\chi\omega_0\frac{dx}{dt} + \omega_0^2 x = 0, \tag{2.8}$$

where χ is the damping ratio and ω_0 is the characteristic frequency. This describes the physics that comprise a range real systems, with the common example being a spring based system that includes friction. In this case, we find that $\chi = \frac{c}{2\sqrt{mk}}$ and $\omega_0 = \sqrt{\frac{k}{m}}$, where m is the mass of the moving body, k is the spring constant, and c is the coefficient of friction. For this study, the values of 0.75, 4, and 0.1 were chosen for m, k, and c, respectively, and used as a standard throughout. Figure 2.6 illustrates how the waveform for this may look.

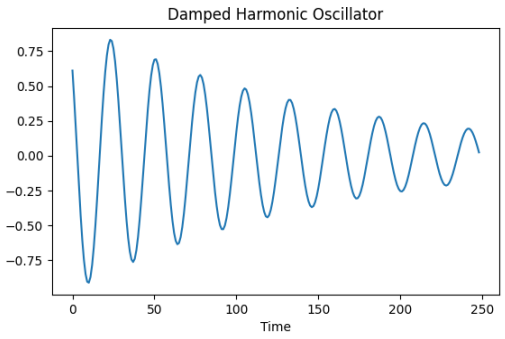


Figure 2.6 *Damped harmonic oscillator* function.

2.5 Fundamentals of Quantum Computing

The fundamental difference between quantum and classical computing is the use of what is known as a "qubit" or quantum bit for the computational object. Each single qubit's state can be represented by a two dimensional vector; a common short hand for this is Dirac notation, where $|x\rangle$ ("ket x") represents state x's column vector and $\langle x|$ ("bra x") represents its row vector, such that for the computational basis,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ \langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix}, \ \langle 1| = \begin{bmatrix} 0 & 1 \end{bmatrix}. \tag{2.9}$$

This manner of linearization is where the superposition of states is encapsulated by quantum computing. To be more concise,

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \beta \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \alpha |0\rangle + \beta |1\rangle . \tag{2.10}$$

Here, α and β are referred to as the amplitudes for the $|0\rangle$ and $|1\rangle$ computational basis states and are complex numbers. Without loss of generality, quantum states like $|\psi\rangle$ should be unit vectors in the complex vector space, where they are constrained by the normalization condition such that

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.11}$$

This is also extended to multi-qubit systems, such that

$$\sum_i |\alpha_i|^2 = 1. \tag{2.12}$$

Unlike reading the state of classical bits which may be done at any time with no consequence, the quantum mechanical nature of qubits prevents us from reading their precise state in the same manner. Regardless of the complex orientation of the qubits, a measurement operation will probabilistically collapse the system to one of the basis states for the measurement, effectively destroying its previous quantum state. For example, if a single qubit were measured in the computational basis then its resulting state will become either $|0\rangle$ or $|1\rangle$. Given this, it should be easy to see from the above equations that the square magnitude of a state's complex amplitude can be interpreted as the probability of observing the quantum computer in the corresponding state.

A useful representation for qubits is the Bloch sphere; here the computational basis states rest at opposite ends of the z-axis on a unit sphere. All other states can be defined as a superposition for some x- and y-axis orientation on that unit sphere based on the state's complex amplitudes. The exact configuration for any given state can also be described in terms of its polar coordinates θ and φ from the z-axis.

In order to have a functional quantum computer, a lot of linear algebra formalism is required. Glossing over the mathematical details in the nature of presenting the fundamentals, quantum computation requires at its core for operations on qubits to be both reversible and unitary. While the definition for a reversible circuit should be implicit by its name, if an operator U is said to be unitary then its conjugate adjoint is equal to its inverse such that $U^* = U^{-1}$. This gives that $UU^* = U^*U = I$, which is useful for a number of reasons not stated here. Similar to the qubit state vectors, operators on n-qubit systems can be represented as a 2^n dimensional square matrix. When an operator is encountered by a qubit passing through a quantum system, the

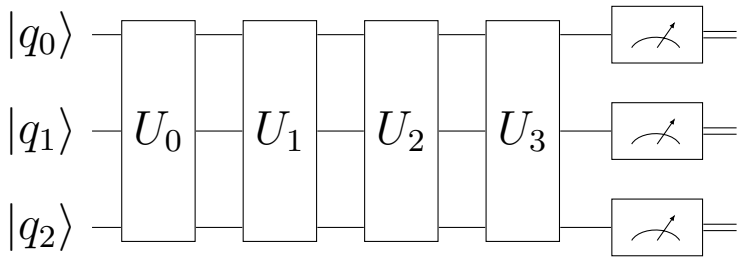


Figure 2.7 A generic quantum circuit that uses 3 qubits and 4 unitary gates, with a measurement performed after the last unitary.

operator acts as a linear map that takes the initial state to a resulting state (e.g. $U|\psi\rangle = |\phi\rangle$) [44, 34, 59].

The presently preferred method for programming a quantum computation consists of building a circuit from unitary operator "gates" representing the mathematical operations being implemented. A generalized example of what such a circuit may graphically look like is shown in Figure 2.7. Although the math must still be understood to produce valid computations, there now exists software development kits such as IBM's Qiskit [1] and Xanadu's PennyLane [7] packages to aid in this process and even queue jobs to the cloud to be run on actual quantum computers. With an intuitive syntax and a graphical circuit builder, Qiskit streamlines the task of implementing and simulating quantum algorithm. Similarly, PennyLane exists as a platform for accelerating the development of quantum circuits through differentiable programming.

Chapter 3

Methods

With the case study details established, it is possible to analyze the performance of a well-known ML model. This provides a baseline with which a QML model of the same variety may be compared. In order to achieve the aforementioned goal of this work, however, a baseline comparison between classical and quantum models alone is not sufficient. Furthermore, the current state-of-the-art prevents a fully quantum implementation from being achievable. This chapter therefore serves to introduce the methods employed to realize the quantum and stochastic model versions investigated. First, Section 3.1 details the principles of VQAs and how they may be applied to implement a QLSTM model. Next, Section 3.2 describes a technique used for quantizing neural network weights to low-precision representations. Finally, Section 3.3 introduces techniques for incorporating stochasticity into ML models.

3.1 Variational Quantum Algorithms

As mentioned in Chapter 1, modern NISQ devices comprise the corpus of currently available quantum computational technology. While there are indeed many working implementations, none are perfect. The main issues that typically arise in all of these cases are due to the quantum system's scalability and decoherence times. For reference, included in Table 3.1 is approximations for decoherence time τ_Q, operation time τ_{op}, and maximum number of operations before decoherence $n_{op} = \lambda^{-1} = \frac{\tau_Q}{\tau_{op}}$ for several physical realizations of qubits [44]. Furthermore, current devices range in size from 50 to 100 physical qubits; with quantum error correction translating this to a very small number of usable logical qubits, the scope of what can be implemented is severely limited.

.
Table 3.1 Approximate decoherence and operational times for various implementations of qubits. (reproduced from [44])

Hardware Implementation Statistics				
System	Qubit Representation	τ_Q	τ_{op}	$n_{op} = \frac{\tau_Q}{\tau_{op}}$
nuclear spin	spin	$10^{-1} - 10^{8}$	$10^{-3} - 10^{-6}$	$10^{5} - 10^{14}$
electron spin	spin	10^{-3}	10^{-7}	10^{4}
ion trap - In^{+}	spin flips	10^{-1}	10^{-14}	10^{13}
electron - Au	charge	10^{-8}	10^{-14}	10^{6}
electron - GaAs	charge	10^{-10}	10^{-13}	10^{3}
quantum dot	spin/charge	10^{-6}	10^{-9}	10^{3}
optical cavity	photon	10^{-5}	10^{-14}	10^{9}
microwave cavity	photon	10^{0}	10^{-4}	10^{4}

Researchers have found, however, that by leveraging quantum circuits with tunable parameters a great majority of the envisioned uses for quantum computing can be achieved. Referred to as variational quantum algorithms (VQAs), these systems typically exist as hybrid systems and use classical optimization techniques to train the circuit parameters. Optimization methods such as those described in Section 2.3 work for this purpose, provided that there is a valid cost function for the problem

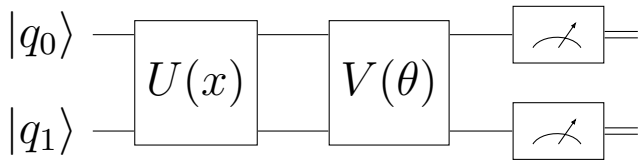


Figure 3.1 A general example of a variational quantum algorithm circuit.

being solved [41, 53, 39]. As such, these VQAs are composed of three primary components: an ansatz suited to solving the task that has been encoded, an optimization procedure over the tunable parameters' gradients, and a cost function that's effective for the problem.

The most crucial portion of this is the ansatz, which consists of a sequence of quantum operations that are either proposed using information about the problem or formulated to be as general as possible. This comprises the body of the quantum circuit for the VQA, defining its structure and encoding the parameters to be trained. Such a construction allows the quantum circuit depth to be kept shallow, making it extremely suitable to NISQ-era devices. While they have shown promise even at a shallow depths, it should noted that the theory of the field suggests that as many unitary gates for data encoding and the primary variational ansatz as necessary can be applied. A very general example of what such a circuit for a VQA may look is illustrated in Figure 3.1; here the data x is first encoded by unitary gate U, then the variational ansatz is applied through unitary V with the parameter θ. After the measurement operation, the parameter will undergo classical optimization with the selected optimizer and cost function [16, 17, 58].

Some applications for this technique include finding ground and excited states of molecules, classical optimization procedures, quantum program compilation, and even quantum error correction, among many others [16, 58]. Given the method's formulation, its easy to see how it can be considered as an analog to classical ML; thus, its natural to consider extending it to such problems in data science, which

is the use case for the work at hand. Research in this area has shown that QML models which utilize VQAs are successful in a variety of problems, can have a greater expressive power, converge to optimal solutions faster, and can even produce better results [11, 17, 68]. Given that ML is a technique used in analyzing data, VQAs used for QML commonly feature an input data encoding layer not unlike that which is used to encode the tunable parameters.

As demonstrated by Chen, Yoo, and Fang [17], a quantum version of the LSTM cell architecture can be realized in a very similar manner to its classical representation. The primary difference is that the MAC operations within the LSTM cell have been replaced by VQA circuits (VQCs). The circuit design encodes the data for the LSTM time step, then applies an ansatz based on entangled state rotations with the current parameters. This is followed by measurements on every wire, where the output is returned to proceed as predestined through LSTM cell's remaining operations. The authors of the work [17] also applied these circuits onto the hidden state passed to the next time step and the network cell output, however investigation done herein showed that this is not necessary for the QLSTM model to achieve high performance. The QLSTM cell then can be illustrated by Figure 3.2, and the LSTM cell equations then become as follows:

$$f_t = \sigma(VQC_1(v_t : \theta_f)) \tag{3.1a}$$

$$i_t = \sigma(VQC_2(v_t : \theta_i)) \tag{3.1b}$$

$$\tilde{C}_t = tanh(VQC_3(v_t : \theta_{\tilde{C}})) \tag{3.1c}$$

$$c_t = (f_t * c_{t-1}) + (i_t * \tilde{C}_t) \tag{3.1d}$$

$$o_t = \sigma(VQC_4(v_t : \theta_o)) \tag{3.1e}$$

$$h_t = o_t * tanh(c_t) \tag{3.1f}$$

The VQC utilized for this problem has an architecture as shown by Figure 3.3. This design calls for the use of four qubits and utilizes three rotation angles, α, β,

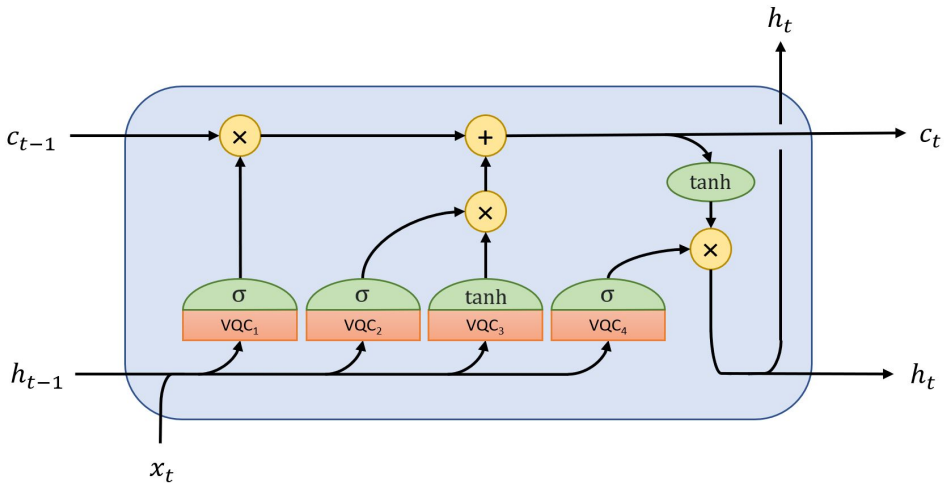


Figure 3.2 Graphical representation of a QLSTM cell.

and γ, for the parameters θ. It consists of three portions: a data encoding layer, a variational layer, and a measurement layer. The circuit starts by placing the initial ground state qubits into an unbiased state via a bank of Hadamard gates. The input data is then encoded as rotational angles for the qubits after transforming its values using the inverse tangent function, first by applying the R_y gate with the trigonometric results and then by applying the R_z gate with the trigonometric results of the data's square. Next, the variational layer generates multi-qubit entanglement through rings of CNOT gates. Following this, the tunable parameters of each qubit are encoded using single qubit rotational gates, i.e. $R(\alpha, \beta, \gamma)$. This variational layer may be applied multiple times, defining a depth hyperparameter to the circuit; Chen, Yoo, and Fang [17] utilize a depth of two, but a depth of only one has been used for this work. As with any quantum circuit we wish to know the results of, the final portion of the circuit then performs measurement operations on each qubit, reading out their state in the computational basis as Pauli-Z expectation values.

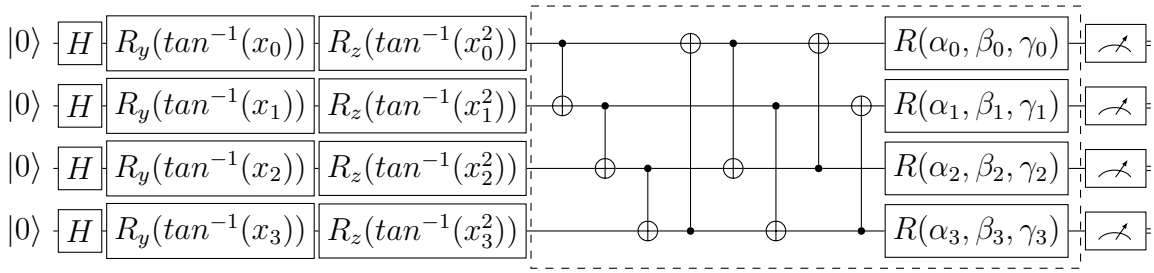


Figure 3.3 The QLSTM VQA circuit architecture introduced Chen, Yoo, and Fang [17]; the variational ansatz in the dashed box can be applied as many times as necessary.

3.2 Weight Quantization

Weight quantization is a technique that has primarily been utilized for deploying ML models on low-precision devices that have limited computing resources and memory capacity, such as embedded systems and mobile devices like cell phones [32, 26, 43, 31, 19, 38, 49, 67, 36, 22]. The data used by these systems is typically constrained to a smaller number of constituent bits than the "real valued" high-precision data allowed by standard computing hardware. Furthermore, floating-point arithmetic may not even be possible on some of these devices. Quantization thus functions to logically reduce data bitwidth, as opposed to simply truncating the excess bits. These techniques generally convert floating-point representations to either fixed-point numbers or discrete integers; for the present work, we will only consider the case where resulting values are integers. Applications for this method can be broadly categorized into cases where a pre-trained model is quantized and cases where a model is quantized during its training, with the latter being the focus for this work.

There are a few quantization methods that have proven effective, although naïve approaches using a small number of resulting bits tend to produce poor results. This is largely due to small updates during optimization being rounded off, causing training to stagnate; this is similar to the vanishing gradient issue in training full-precision models, and can effectively have the same impact on optimization. The two methods

introduced here are common choices when applying weight quantization, those being deterministic rounding and stochastic rounding. While the former of these options is more prone to aforementioned issue, stochastic rounding is less affected by it given the method's probabilistic nature. The functions for these are presented in Equations 3.2 and 3.3, where w is the weight to be quantized, Q_d is deterministic rounding, Q_s is stochastic rounding, and p is a uniformly generated random number. These functions are commonly followed by a clipping, or clamping, function to ensure the result is within the (signed or unsigned) range specified by the bitwidth.

$$Q_d(w) = round(w) \tag{3.2}$$

$$Q_s(w) = \begin{cases} \lfloor w \rfloor + 1, & \text{for } p \leq w - \lfloor w \rfloor \\ \lfloor w \rfloor, & \text{otherwise} \end{cases} \tag{3.3}$$

Although the option may not be available on low-precision devices, many researchers have suggested that maintaining a high-precision floating-point copy of quantized weights during training can improve model results. This strategy involves performing the feed-forward and backprop operations with quantized weights then applying the update rule to the high-precision weights, before quantizing the result for the next training iteration. When applied to standard gradient descent, this has the form:

$$w_{q,t+1} = Q(w_{r,t+1}) = Q\left(w_{r,t} - \alpha \frac{\delta E}{\delta w_{r,t}}\right) , \tag{3.4}$$

where $w_{q,t}$ indicates the quantized weight at time t, $w_{r,t}$ is the real weights at time t, and $Q(\cdot)$ is the quantization function. This can be extended to any optimization routine by altering the rightmost expression to use the proper update procedure prior to application of the quantization function. Its worth noting before proceeding that quantization is typically applied only a model's weights and not to its biases. [37, 18].

3.3 Stochastic Techniques

To accomplish the intended goal stated in Chapter 1, it is necessary to imbue the model investigated with stochastic techniques. One possible approach for including stochasticity is the use of binary stochastic neurons within the network [2]. Owing its name to their binary and stochastic output, this type of neuron utilizes probabilistic activation functions and produces one of two possible outcomes. While keeping MAC operations normal, these constructs will probabilistically determine their result to be either the maximum or minimum of the original activation function by utilizing its result as a probability distribution. As such, a binary stochastic neuron using sigmoid will either have the outcome 1 or 0, and the likelihood of seeing either is directly determined by its standard sigmoid activation, given that its output is already in the range $[0, 1]$. This is mathematically represented below by the set of probabilities:

$$P_\sigma(1) = \sigma(W \cdot v_t + b) \tag{3.5a}$$

$$P_\sigma(0) = 1 - P(1) = 1 - \sigma(W \cdot v_t + b) \tag{3.5b}$$

This concept is fairly straightforward for the sigmoid activation function; however, for an activation like the hyperbolic tangent that has a different output range, the probabilities are slightly more complicated. While still a trivial matter, its original output must be transformed from the range $[-1, 1]$ to values in the range $[0, 1]$ in order to be interpreted as a probability. Therefore, a binary stochastic neuron utilizing hyperbolic tangent can be realized with the following probabilities:

$$P_{tanh}(1) = \frac{tanh(W \cdot v_t + b)}{2} + 0.5 \tag{3.6a}$$

$$P_{tanh}(-1) = 1 - P(1) = 0.5 - \frac{tanh(W \cdot v_t + b)}{2} \tag{3.6b}$$

Another potential strategy for incorporating stochasticity can be realized through the use of stochastic rounding as discussed in the previous section and demonstrated by Equation 3.3. However, the stochastic benefits of this approach in its commonly

applied form are limited to the training phase of a model. Despite this, quantization can be repurposed to incorporate stochasticity into the inference phase by generalizing its application beyond just model weights. While an uncommon practice, the same method described in Section 3.2 can be applied to the output of MAC operations within ML model layers. This approach can be mathematically described by reworking Equation 3.4 as

$$\hat{u_t} = Q(u_t) = Q(W \cdot v_t + b) \ , \tag{3.7}$$

where u_t refers to the MAC output before application of a quantization function, $\hat{u_t}$ is the quantized MAC output, and W and b are respectively the set of weights and biases for the MAC operation. By using Equation 3.3 as the quantization function, stochasticity can be effectively incorporated into the model's feed-forward path and as such within its inference phase.

Chapter 4

Results and Analysis

Having formed the necessary basis for this work with the case study and methods in Chapters 2 and 3, we can now explore the analyses performed and results gathered. The models discussed herein are all based on the LSTM architecture and use the hyperparameters listed in Table 4.1 as a standard, unless otherwise noted. All implementation is done and tested using Python 3.8.10. Throughout the following sections, figures, and tables, the "best epoch" is defined as the epoch with the lowest validation RMSE metric value. The data and figures relevant to the discussion of this chapter are included herein, while the complete set of graphics for all experiments can be found in Appendices A-E.

Table 4.1 Hyperparameters used throughout experimentation.

Hyperparameter Settings	
Setting	Value
input dimension	1
hidden dimension	5
output dimension	1
batch size	4
test set split size	0.33
learning rate	0.01
ρ (for RMSProp)	0.99
ϵ (for RMSProp)	1e-8
training epochs	100

First in Section 4.1, the foundations set by the standard or classical LSTM model implemented in-house and used in subsequent versions is discussed. Then Section

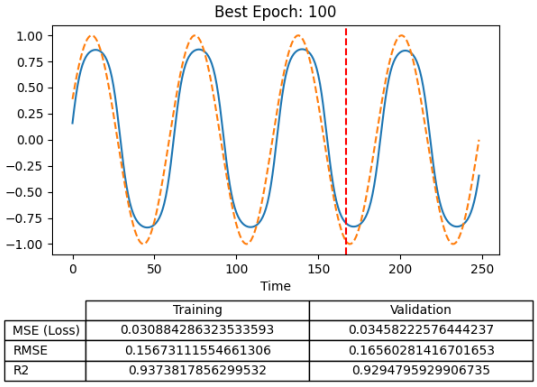
4.2 presents the quantum version of the model and its performance. Following this, Section 4.3 investigates a common classical application of stochasticity by analyzing an LSTM model utilizing weight quantization methods. Section 4.4 then shows the initial results achieved with models using the stochastic methods previously described in Section 3.3. Next, Section 4.5 highlights specific results dependent upon quantum device specifications, before Section 4.6 details alternate versions of the stochastic models using a modified scheme inspired by the preceding section. Finally, Section 4.7 exhibits the full breadth of results gathered by this work and ultimately attempts to answer the question proposed in Chapter 1: **Can we achieve similar results to variational QML by incorporating stochasticity inspired by its architectures into classical machine learning models?**

4.1 In-House LSTM Model

While the development of a classical LSTM model plays a minor role in the major findings of this work, it was inherently necessary to use an in-house implementation in creation of the stochastic models featured in the coming sections. If the only concern was comparing the performance of QML with classical ML, then a model developed with industry-standard libraries like TensorFlow or PyTorch would have sufficed, such as done by Chen, Yoo, and Fang [17]. Without a comprehensive knowledge of the back-end for such a library, however, it becomes extremely difficult to implement the sought after stochastic techniques. By implementing the network from the ground up and through direct modifications, it became possible to alter model functionality to a much finer granularity.

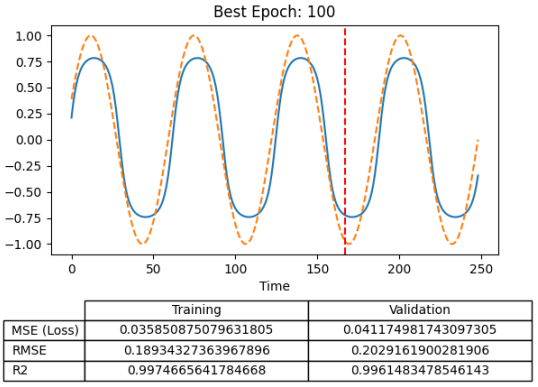
Given this, construction of an in-house model was the starting point for the work at hand. The model implemented was realized through three separate Python classes: one for the LSTM parameters, one for the LSTM cell, and one for the LSTM network architecture. These classes are supported by several utility methods, implementing functions such as optimization, metrics, and activations, in addition to a driver program that enables the model to be utilized for experiments.

The first aforementioned class primarily exists to contain the weights and biases to be utilized by all LSTM cells. The second class forms the LSTM cell itself, with Equations 2.1 building the feed-forward algorithm for the node. Back-propagation as described in Section 2.3 is also achieved here; using manually derived equations, the node receives loss derivatives with respect to the cell and hidden states, propagates them using the chain rule, and individually accumulates the loss derivatives with respect to the weights and biases before returning the loss derivatives with respect to the cell and hidden states for the previous time step. The third class mentioned constructs the LSTM network architecture, creating a sequential list of cells that comprise the "unrolled" form of the model. It is also here that the training and



	Training	Validation
MSE (Loss)	0.030884286323533593	0.03458222576444237
RMSE	0.15673111554661306	0.16560281416701653
R2	0.9373817856299532	0.9294795929906735

(a)



	Training	Validation
MSE (Loss)	0.035850875079631805	0.041174981743097305
RMSE	0.18934327363967896	0.2029161900281906
R2	0.9974665641784668	0.9961483478546143

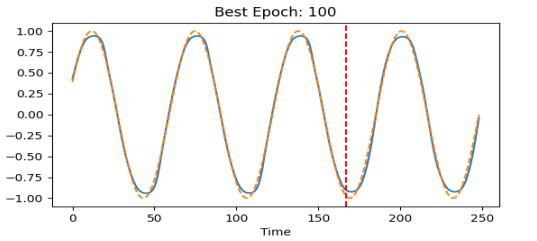
(b)

Figure 4.1 Best epoch data of LSTM models optimized by *standard gradient descent* for *sine* function. (a) in-house implementation, (b) TensorFlow implementation.

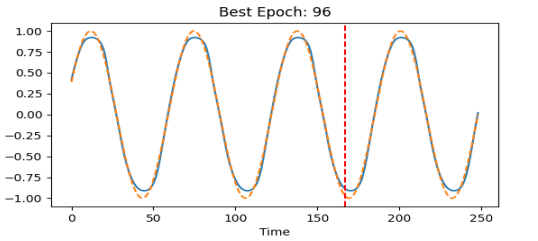
inference procedures are defined, following the standard format. It should be noted that for all experiments and subsequent model versions, the final LSTM cell's output is passed to a fully-connected layer using a hyperbolic tangent activation function; this essentially transforms the model's hidden dimension to its output dimension.

In order to verify the validity of this in-house model, the same LSTM network was implemented using the TensorFlow library. Using the *sine* periodic function as the baseline experiment, the in-house model produced results comparable to that achieved by the TensorFlow model. In our experiments, the in-house model slightly outperformed the TensorFlow model, producing a validation RMSE of 0.1656 while the library implementation produced a value of 0.2029 in the corresponding best epochs; the results of this test are shown in Figure 4.1.

Although the R^2 it achieved was slightly lower than the out-of-the-box counterpart, the margin of error is close enough to consider the in-house model as successfully verified. Despite this, both models produce less than desirable results for the experiment presented in Section 2.4 most commonly used to benchmark LSTM implementations. This can be explained, however, by the use of *standard gradient descent* optimization in both cases; this is the most common, and perhaps basic, form of



	Training	Validation
MSE (Loss)	0.0015187249160366519	0.003293368874945428
RMSE	0.03148512911492807	0.05001459463383491
R2	0.9969207693075605	0.9932841305509086



	Training	Validation
MSE (Loss)	0.0030444045551121235	0.0024955265689641237
RMSE	0.05517612397670746	0.04995524510741234
R2	0.9998880624771118	0.9997047781944275

(a) (b)

Figure 4.2 Best epoch data of LSTM models optimized by *RMSProp* for *sine* function. (a) in-house implementation, (b) TensorFlow implementation.

gradient optimization used in modern ML.

Therefore, the logical solution to improving the model efficacy was to implement and utilize a more sophisticated optimization method. Using the *RMSProp* optimization procedure discussed in Section 2.3 and with hyperparameters listed in Table 4.1, both models achieved much better results. The best epoch data is visualized in Figure 4.2 for this test. With the differences in RMSE and R^2 values being +0.001 and -0.006, respectively, the in-house implementation slightly underperformed compared to the standard library model; however, it is within enough of an acceptable margin of error to be considered successful. To the human eye, the plots included in Figures 4.2a and 4.2b look nearly identical graphically, with only minor variations at the predicted waveform peaks.

With this, the in-house LSTM model can be considered as successfully implemented and verified. Due to the nature of improvement in model performance for the *sine* function over that achieved by using the *standard gradient descent* optimizer, the *RMSProp* optimizer was employed for the remaining in-house model experiments. The best epoch metric data for all experiments is presented in Table 4.2; the only function with an outlier performance was the *sawtooth* function, with a relatively high MSE value of 0.0769 and correspondingly low R^2 value of 0.787. However, inspecting

the waveform predicted in the best epoch for this function, included in Appendix A.5 but not shown here, shows that the errors are primarily introduced around the peaks.

Table 4.2 Metrics from the best epoch for the in-house classical model over all experiments.

LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0015	0.0315	0.9969	0.0032	0.0500	0.9933
Sawtooth	0.0416	0.0484	0.8593	0.0769	0.0905	0.7870
Summed Waves	0.0051	0.0564	0.9797	0.0059	0.0625	0.9761
Oscillator	0.0004	0.0140	0.9978	0.0005	0.0199	0.9842

This makes it apparent that the error in prediction is the result of the function being non-continuous; it is possible that the simultaneously linear and asymptotic nature of the function is difficult to learn with the configurations standardized in Table 4.1. Rather than experimenting to find unique settings that could produce a better performance, this has been deemed the acceptable level of performance for this function throughout the work. Thus, the success demonstrated by these results when utilizing the *RMSProp* optimizer gives reasonable grounds to adopt it as the standard for this study. As such, it is employed by all subsequent model versions reported herein.

4.2 Quantum LSTM Model

Considering the in-house model's construction described in the previous section to be this work's case-study basis, its core focus and backbone was still missing. Thus, implementation of a quantum version of the LSTM model, using techniques previously described in Sections 2.5 and 3.1, became imperative. Thankfully, the work performed by Chen, Yoo, and Fang [17] existed as a precursor to this effort and provided directions to pursue. The desired model needed to utilize the VQA circuit shown in

Figure 3.3 with the LSTM cell architecture of Figure 3.2. Although formulating such a model manually on a hybrid classical-quantum computational system would be no simple feat, application of the previously described PennyLane [7] library enabled a relatively straightforward approach to this problem.

In what could be considered a quantum analogue of automatic differentiation libraries like TensorFlow and PyTorch, PennyLane provides nearly everything off the shelf needed to construct hybrid classical-quantum systems. However, the most important features to this work it contains can be stated as its ability to instantiate quantum devices, its range of pre-built quantum operator methods, and its use of QNode objects to convert quantum circuits into nodes within hybrid computational graphs.

Arguably the most important part of quantum computational systems, the devices are where quantum circuits are built and run. Although modern quantum hardware is scarcely available, PennyLane provides not only a simulator to run circuits on but also interfaces that make most current cloud quantum backends accessible. Moreover, its suite of operator methods makes development of quantum circuits a simple matter when given the desired architecture. Once both a device and circuit has been constructed, the QNode object essentially binds the two together before inserting itself as a node within the hybrid computational graph for the system [7].

With PennyLane incorporating the quantum portion of the hybrid model, the remaining task for an end-to-end system was to build its classical portion. Given the success with implementing the in-house model, its natural to assume this would be a simple matter; however, the interoperability between classical and quantum systems existed as the major difficulty for the construction of this model. Attempts to integrate the in-house model with the PennyLane architecture were met with consecutive and undocumented program bugs, and utilization of TensorFlow proved fruitless in the same manner. Capitalizing on the precedence set by Chen, Yoo, and Fang [17]

once more and implementing the hybrid model with PyTorch, however, managed to resolve the terminal issues otherwise exhibited. With the successful hybridization of PennyLane and PyTorch libraries to form a LSTM architecture, the quantum version of the model was ready to be deployed.

All experiments performed using the QLSTM model utilized 4 qubits for quantum circuits, only 1 variational layer, and the "default.qubit" simulator device. It was desired to also perform the experiments using IBM's cloud quantum backend as a device, but the lack of a reliable server with the correct dependencies needed to deploy the model kept this goal from coming to fruition. Such a server was deemed necessary for this task due the extremely long wait times to complete a cloud quantum job, as a result of the queues for device access.

As expected based on existing literature, every experiment run on this model produced exceptional results. The best epochs' metrics for all tests are recorded in Table 4.3. Comparing this with the metric data reported in Table 4.2, its clear that both models are reaching nearly the same levels of performance. Its interesting to note that the quantum model has better performance metric values than the classical model for all experiments, except for the *sine* function. This upshot in performance could indicate that the in-house model performs better on *sine*, but it would be remiss to not consider that this might be explained by a stronger initialization.

Table 4.3 Metrics from the best epoch for the quantum model over all experiments.

QLSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0028	0.0408	0.9944	0.0042	0.0554	0.9914
Sawtooth	0.0407	0.0465	0.8624	0.0727	0.0907	0.7989
Oscillator	0.0008	0.0120	0.9959	3.2e-5	0.0049	0.9990
Summed Waves	0.0021	0.0318	0.9917	0.0045	0.0559	0.9817

Regardless of the performance achieved for the individual experiments, all trials

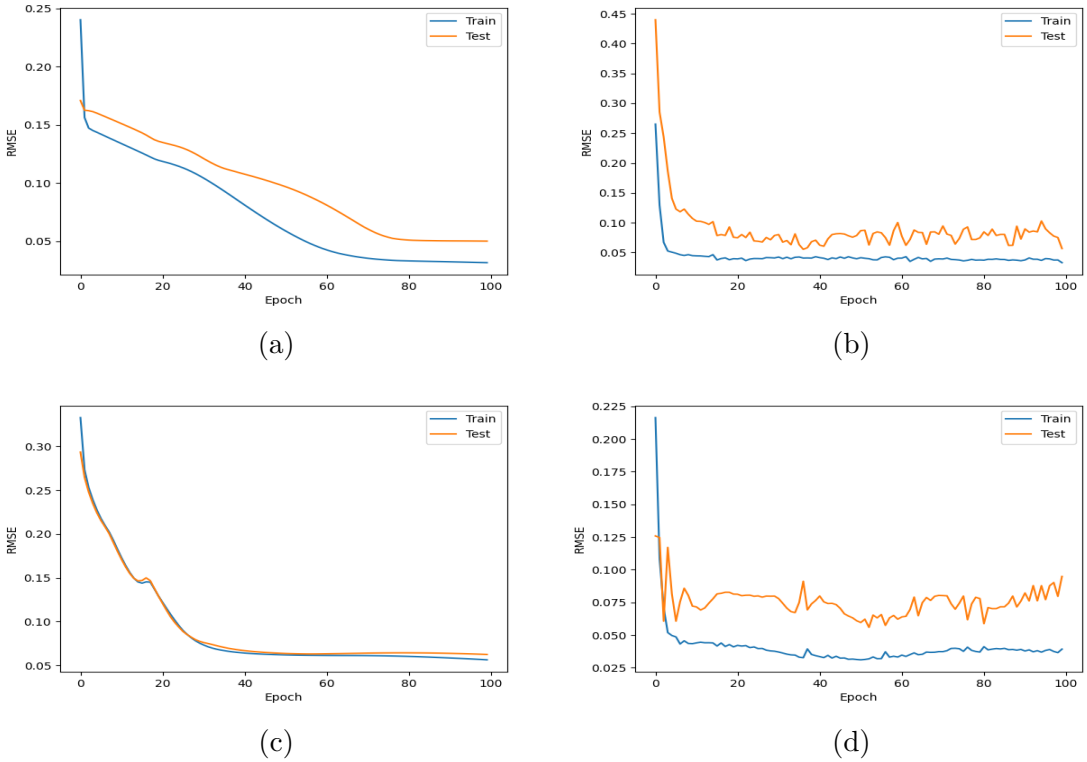


Figure 4.3 RMSE metric data from *sine* and *summed waves* functions for the in-house and quantum LSTM models. (a) *sine* on in-house model, (b) *sine* on quantum model, (c) *summed waves* on in-house model, (d) *summed waves* on quantum model.

appear to converge much faster than they did with the classical model. This is visualized in Figure 4.3, where the RMSE metric graphs for the *sine* and *summed cosine waves* functions on both models are shown together. By about the 5th epoch, the quantum results have converged to a minima. Following this, the metrics illustrate fluctuations in model performance for the remaining epochs. This is likely the result of the algorithm hunting for the global minima. On the other hand, the classical results converge more steadily over longer time frame. This continuous and gradual convergence allows the model to continues approaching its optimum with minimal fluctuations. The nature of optimization that this observation appears to impart could be described such that the in-house model is "learning the optimum" while the quantum model is "searching for the optimum."

4.3 Quantized Model

With the classical and quantum models established, we can now begin to investigate probabilistic ML models. The first direction taken was to give consideration to the limited number of qubits available for quantum models and how a measurement operation collapses their individual states to either $|0\rangle$ or $|1\rangle$, as mentioned in Section 2.5. With this in mind, it seemed that utilizing a model with quantized weights as described in Section 3.2 would potentially provide a fairer comparison than the full-precision and floating-point classical model with regard to the advantages leveraged by QML. As an additional benefit, use of the stochastic rounding method would present a way to incorporate stochasticity within the model.

Realizing an implementation for the quantized model was fairly simple with the in-house model already available. After implementing Equations 3.2 and 3.3 along with updates to model parameters, weight initialization, feed-forward operation, and optimizer, its construction was ready for testing. It is worth noting that results for both quantization methods reported in this section use the same initialization for the respective experiments. Additionally, a bitwidth of 4 was utilized for all trials, giving signed integer range of [-8, 7] for weight values.

Regarding the performance of only the quantized models as reported in Tables 4.4 and 4.5 for the moment, it appears that stochastic rounding is capable of outperforming the alternative presented. Although the deterministic rounding variant achieved better training metrics for the best epoch recorded, it is surpassed during validation for all experiments used. This is clearly illustrated in Figure 4.4, which shows the RMSE metric data of the two methods plotted together. While the training performance for both is relatively stable in convergence, the validation performance generally fluctuates between epochs more for stochastic rounding. This is expected; the method's probabilistic nature allows it to more flexibly navigate the loss landscape, but may also move the model away from the optimal configuration. Thus, some

training iterations may be negatively impacted while still being capable of overcoming cases where the deterministic version might stagnate.

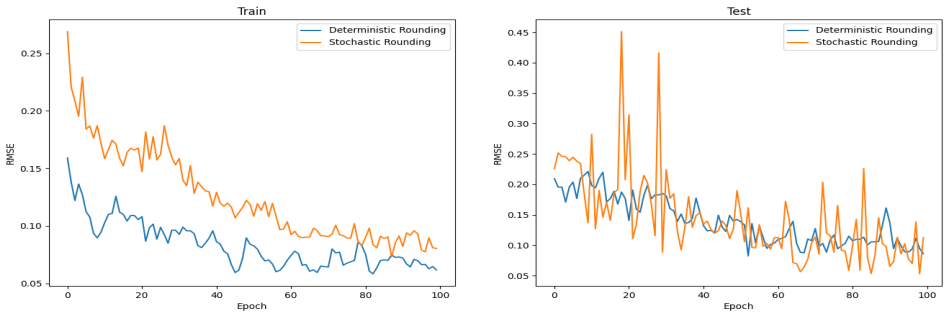
Table 4.4 Metrics from the best epoch for the deterministic rounding based quantized model over all experiments.

Deterministic Rounding Quantized LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0090	0.0735	0.9818	0.0104	0.0823	0.9788
Sawtooth	0.0498	0.0655	0.8316	0.0701	0.1173	0.8060
Summed Waves	0.0227	0.1160	0.9092	0.0196	0.1104	0.9206
Oscillator	0.0033	0.0434	0.9825	0.0010	0.0266	0.9690

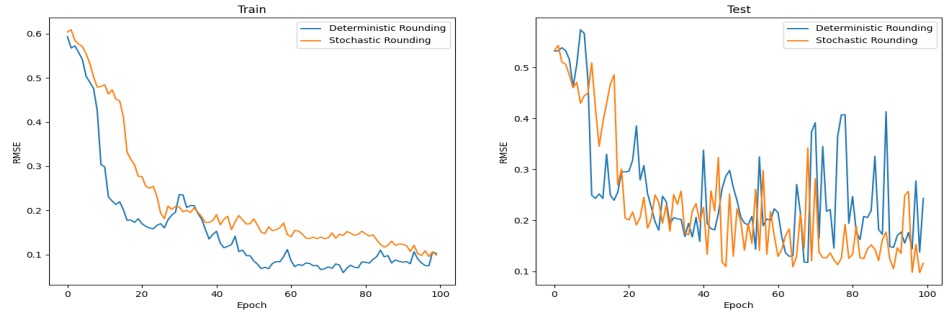
Table 4.5 Metrics from the best epoch for the stochastic rounding based quantized model over all experiments.

Stochastic Rounding Quantized LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0108	0.0811	0.9781	0.0049	0.0529	0.9900
Sawtooth	0.0512	0.1061	0.8269	0.0823	0.0972	0.7721
Summed Waves	0.0232	0.1209	0.9072	0.0130	0.0912	0.9472
Oscillator	0.0060	0.0585	0.9679	0.0008	0.0199	0.9745

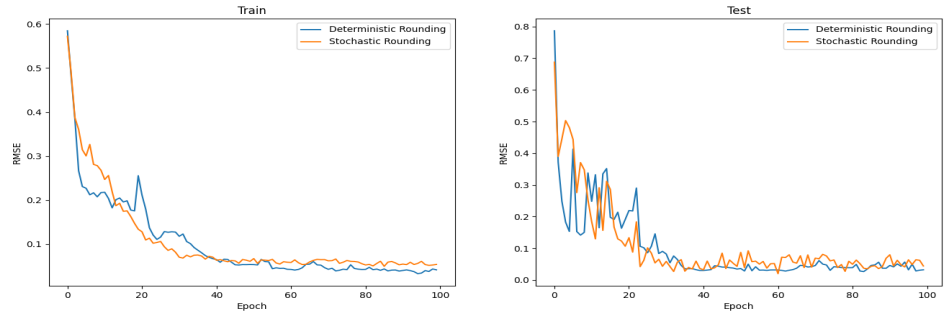
Referring back to the results presented in Sections 4.1 and 4.2, its clear that the deterministic rounding version fall short when compared to both the classical and quantum models. On the other hand, however, the stochastic rounding version performs nearly on par with the classical implementation. While still outclassed by the quantum model, this result is indicative of the potential that stochastic methods possess for improving ML models; use of such a technique here allowed for performance close to a standard construction, even with low-precision parameters that are constrained to integers. Its also worth noting that the convergence speed for both quantization methods appears similar to that of the classical model.



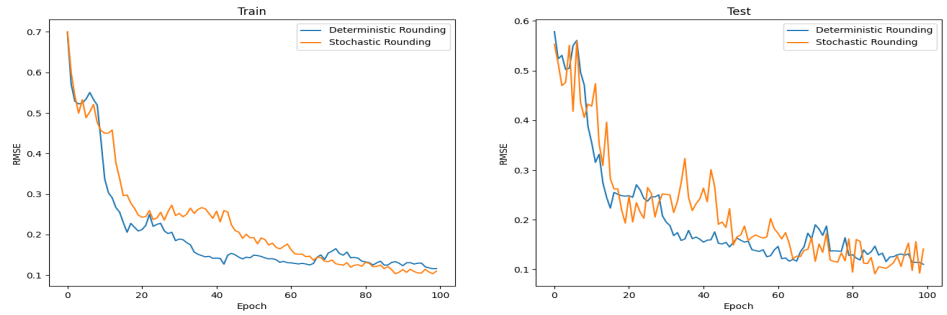
(a) *sine*



(b) *sawtooth*



(c) *damped harmonic oscillator*



(d) *summed waves*

Figure 4.4 RMSE metric data (training on left, validation on right) for both quantization methods.

While these results are promising, its worth giving pause here to consider the initial reasoning previously mentioned for pursuing this model version. The idea was to use an implementation that might be closer to what is logistically occurring in the quantum model, while also introducing a factor of stochasticity. Considering the nature of quantum computing and the QLSTM's VQA circuits, however, this idea may have been misguided; while only a limited number of qubits are usable, their measurement results are used as individual "units" within the QLSTM network rather than as a single low-precision register. Furthermore, the quantum state of each qubit has unrestricted access to the full space of possible tensored complex amplitudes that the paradigm permits.

As an additional point, the stochasticity imbued within the stochastic rounding version of this model fails to capture the essence of stochasticity present within the quantum implementation. By augmenting the optimization procedure with this method, stochasticity is only incorporated as a facet of the training procedure; utilizing a pre-trained model of the same nature for inference would be wholly deterministic. The source of stochasticity in quantum computing lies in the device noise and its measurement operations that collapse the quantum states. As such, a QML model will exhibit stochastic behavior even during inference after its training is completed. Moreover, with a hybrid classical-quantum model, such as the one utilized for this work, the parameter optimization is performed deterministically on the classical portion of the system. Thus, in order to properly analyze the nature of stochasticity within QML systems, a different approach needs to be adopted.

4.4 Stochastic Models

Following the perspective discussed in the previous section, focus for the investigation shifted to methods which would bring elements of stochasticity into not just the training phase but also the network inference. As described in Section 3.3, the use of

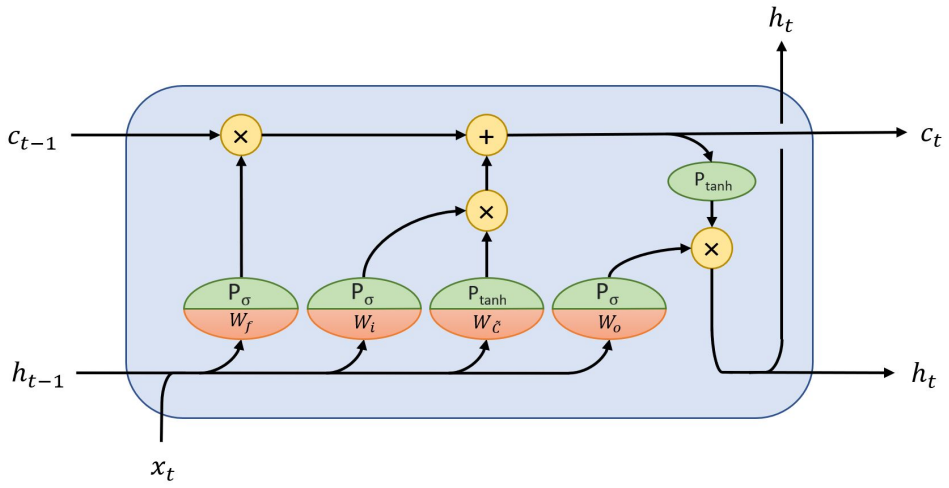


Figure 4.5 Graphical representation of an BSN-based LSTM cell.

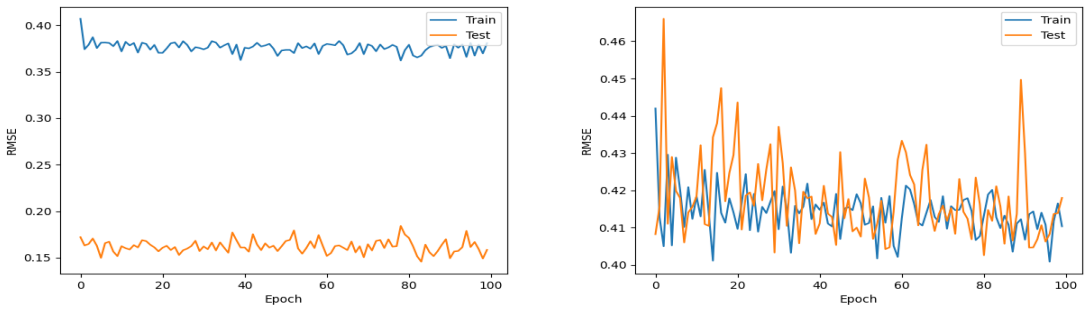
binary stochastic neurons (BSNs) was an attractive option for this purpose. By employing these constructs within the network architecture, stochasticity would become an implicit part of the model. Noting here that the final fully-connected layer of the model still utilizes a standard neuron, Figure 4.5 presents the revised construction of the LSTM cells used by this model version. Implementation of this was achieved with a simple modification to the in-house model to use the probabilistic activation functions described by Equations 3.5 and 3.6 in the feed-forward routine.

Despite seeming to be a good approach conceptually, it would be sufficient to call this model version and endeavor a failure. This is made abundantly clear by the best epoch performance metrics reported in Table 4.6. To help visualize this model's inability to learn the patterns its trained on, the RMSE metric plots for the two datasets that it performed best on, those being the *damped harmonic oscillator* and *summed cosine waves* experiments, are included here in Figure 4.6. It may initially appear from the validation RMSE value for the *damped harmonic oscillator* experiment that the model exhibited minor success in learning; however, this can be explained by the small wave amplitudes of the validation portion of this dataset.

In all cases, its clear that the BSN-based stochastic model was incapable of learn-

Table 4.6 Metrics from the best epoch for the stochastic model utilizing binary stochastic neurons over all experiments.

Binary Stochastic Neuron LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.4735	0.6043	0.0400	0.4358	0.5763	0.1113
Sawtooth	0.2806	0.4410	0.0508	0.3736	0.5147	-0.0342
Summed Waves	0.2597	0.4133	-0.0373	0.2530	0.4025	-0.0253
Oscillator	0.1830	0.3673	0.0192	0.0293	0.1457	0.0573



(a) (b)

Figure 4.6 RMSE metric data from the two functions that the stochastic LSTM using binary stochastic neurons performed best on. (a) *damped harmonic oscillator*, (b) *summed cosine waves*.

ing the data it was fed to any acceptable degree. As further proof of this, some of the R^2 values reported in Table 4.6 are even negative. Despite stochastic techniques typically leading to desirable results, such as demonstrated in the previous section, the BSNs probabilistically forcing their outputs to the extremums of their standard counterparts ultimately caused far more performance degradation to the model than it may have benefited. This would be a case of too much stochasticity. Thus, a different strategy needs to be adopted in order to effectively incorporate stochasticity into the model.

The next attempt to construct a stochastic model draws on the other technique introduced in Section 3.3: quantization of network MAC outputs, specifically with

the use of stochastic rounding. As mentioned, this method is a rather uncommon application of quantization; to this author's knowledge, there are no well-known examples of its use. The idea came about as a result of further consideration on the function of VQA circuits within the quantum model. At the high level conceptually, these circuits essentially replace the MAC operations present within the standard implementation; data is fed into the quantum circuits and their output then receives a non-linear activation. Therefore, any stochasticity exhibited by the quantum model can be considered as a result of its analogue for the standard MAC operation.

Furthermore, the VQA circuits only experience the element of stochasticity during the measurement operation, after its primary quantum operations have been completed. To faithfully imbue a classical model with the same manner of stochasticity, it must be incorporated just after the MAC operations and before application of the relevant activation function. Thus, the utilization of stochastic rounding quantization on MAC output injects an otherwise classical model with a form of stochasticity similar to that experienced by quantum models. The LSTM cell's new configuration using this concept of quantized MAC (QMAC) output is then given as illustrated by Figure 4.7. As with the model versions previously described, the fully connected layer that analyzes the LSTM cell output is left unaltered. A quantization bitwidth of 5 was used for the results reported here to allow the QMAC operations a greater result range than what is possible with the bitwidth used for the weight quantized models.

With the experiment results given in Table 4.7, this QMAC-based model proves to be a far more successful stochastic model than the version using BSNs. Although these are extremely promising results, it is still outperformed by the classical and quantum models, and in some cases one or both of the weight quantized models. Nevertheless, this is still impressive when considering that its inference is non-deterministic by design. If this is the extent of capabilities afforded to classical ML by stochasticity

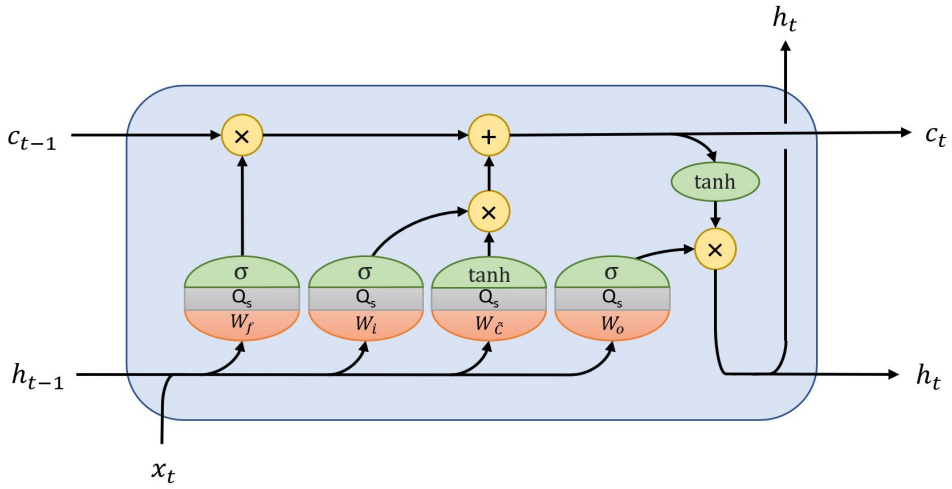


Figure 4.7 Graphical representation of a QMAC-based LSTM cell.

Table 4.7 Metrics from the best epoch for the stochastic model utilizing quantized MAC operations over all experiments.

Quantized MAC LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0084	0.0717	0.9830	0.0079	0.0715	0.9839
Sawtooth	0.0458	0.0819	0.8450	0.0650	0.1029	0.8200
Summed Waves	0.0134	0.1184	0.9119	0.0134	0.0897	0.9455
Oscillator	0.0249	0.1248	0.8665	0.0068	0.0642	0.7810

when compared to QML, then it is hard to consider it as the discipline's source of advantage; however, perhaps there is yet more to the matter which can give a further concrete answer than what can be determined here alone.

4.5 1-Shot Quantum Model

Given the previous section's findings, it would seem that one of two situations must be the case: either stochasticity accounts for a minor amount, if any, of QML's reported advantages, or comprehension of the facet within QML itself requires more investigation. While it would be simple to accept the former attitude, the latter

approach was adopted with the hope to gain a greater insight necessary to properly incorporate stochasticity into classical models. As previously mentioned, the source of this property is determined to be the measurement operations on quantum circuits. This will collapse a qubit's state to one of the basis states for the measurement; for the computational basis, this will be either the $|0\rangle$ or $|1\rangle$ state. Logically, this stochastic outcome would vastly reduce the usefulness of quantum computing as a whole; the mathematical principles that the paradigm evolved from would reduce to nothing more than a form of probabilistic computing. How, then, do quantum algorithms overcome this challenge?

The answer for this is the use of expectation values, otherwise described as an average of all possible outcomes weighted by their probabilities. Application of this concept would then result in the measurements taken giving values in the set of real numbers, rather than only producing a binary result of $|0\rangle$ or $|1\rangle$. Given a quantum system in the state $|\psi\rangle$ and some valid quantum operator U, we can define the expectation value of measuring U as

$$\langle U \rangle_{\psi} := \langle \psi | U | \psi \rangle . \tag{4.1}$$

Here we call U an observable, with the possible results from performing a measurement on U being its eigenvalues.

It may be difficult for the novice to see how this realizes the definition of an expectation value given above; by applying spectral decomposition to view U in terms of its eigenvalues λ_i and eigenvectors $|\phi_i\rangle$, however, this becomes clearer. To be concise, this formalism can be mathematically described as

$$U = \sum_i \lambda_i |\phi_i\rangle \langle \phi_i| . \tag{4.2}$$

Rewriting Equation 4.1 using this formulation of U, we then have

$$\begin{aligned} \langle U \rangle_{\psi} &= \langle \psi | \, U \, | \psi \rangle \, , \\ &= \sum_{i} \lambda_i \, \langle \psi | \phi_i \rangle \, \langle \phi_i | \psi \rangle \, , \\ &= \sum_{i} \lambda_i \, | \, \langle \phi_i | \psi \rangle \, |^2 \, , \end{aligned} \tag{4.3}$$

where $| \langle \phi_i | \psi \rangle |^2$ gives the probability of a measurement resulting in λ_i when the system is in state $|\psi\rangle$.

This provides an analytical method for computing the exact expectation value of a quantum system. As mentioned however, it is impossible to know the precise configuration of a quantum state and a measurement will collapse it to one of the relevant basis states. Therefore, computing the expectation value of a system in practice consists of performing repeated iterations of the quantum circuit and averaging the sum of stochastic measurement results. With enough trials, this experimental solution should converge to the exact expectation produced by the analytical method explained above.

By default, the PennyLane quantum simulator device used in this work will return the exact expectation of measurement operations performed. This means that the results reported in Section 4.2 for the quantum model are in fact utilizing the analytical solution of its measurement operations. As such, they feature no element of stochasticity, even such as that which would be exhibited by running the model on true quantum hardware with enough trials to converge the measurement results.

It is possible, however, to reconfigure the PennyLane simulator to produce stochastic results under the premise presented by experimental expectation values. By specifying the number of "shots" for the device to use, a quantum circuit can be evaluated for a variable amount of times to produce more experimental expectation values. Doing so will enable the stochastic nature of quantum computing, even in a classical simulator. In the following results given in Table 4.8, the previously established quan-

tum model has been tested using the simulator with a single shot for evaluation. In other words, every VQA circuit for the model is run exactly once and the stochastic output from measurements is the data that flows through the model.

Table 4.8 Metrics from the best epoch for the quantum model with 1 shot expectation values over all experiments.

1-Shot Quantum LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0510	0.1729	0.8965	0.0516	0.1718	0.8947
Sawtooth	0.1146	0.2350	0.6124	0.1430	0.2603	0.6041
Summed Waves	0.0734	0.2217	0.7067	0.0635	0.1973	0.7425
Oscillator	0.0338	0.1463	0.8186	0.0122	0.0873	0.6081

Unlike the quantum results of Section 4.2, the performances indicated here are far more comparable to that achieved by the QMAC-based stochastic LSTM of the preceding section. In fact, the QMAC model outperforms the 1-shot quantum model in every experiment tested. Furthermore, the QMAC model appears to more stably converge to its optimum with a similar speed, shown in Figure 4.8 for the *sine* and *summed cosine waves* experiments. While this is certainly impressive, it does not change that the QMAC model still falls short of the analytical quantum model. Perhaps by adapting the technique described in this section, however, the performance achieved by the QMAC model can be further improved.

4.6 Multi-Shot Stochastic Models

Although the quantum model discussed in Section 4.2 uses an analytical solution, the idea of an experimental expectation value poses an interesting potential for stochastic models. If the quantum model can converge from the results using single shot evaluation to that possible with the exact expectation after enough repeated iterations, then stochastic models may be capable of converging in a similar manner. If a formulation

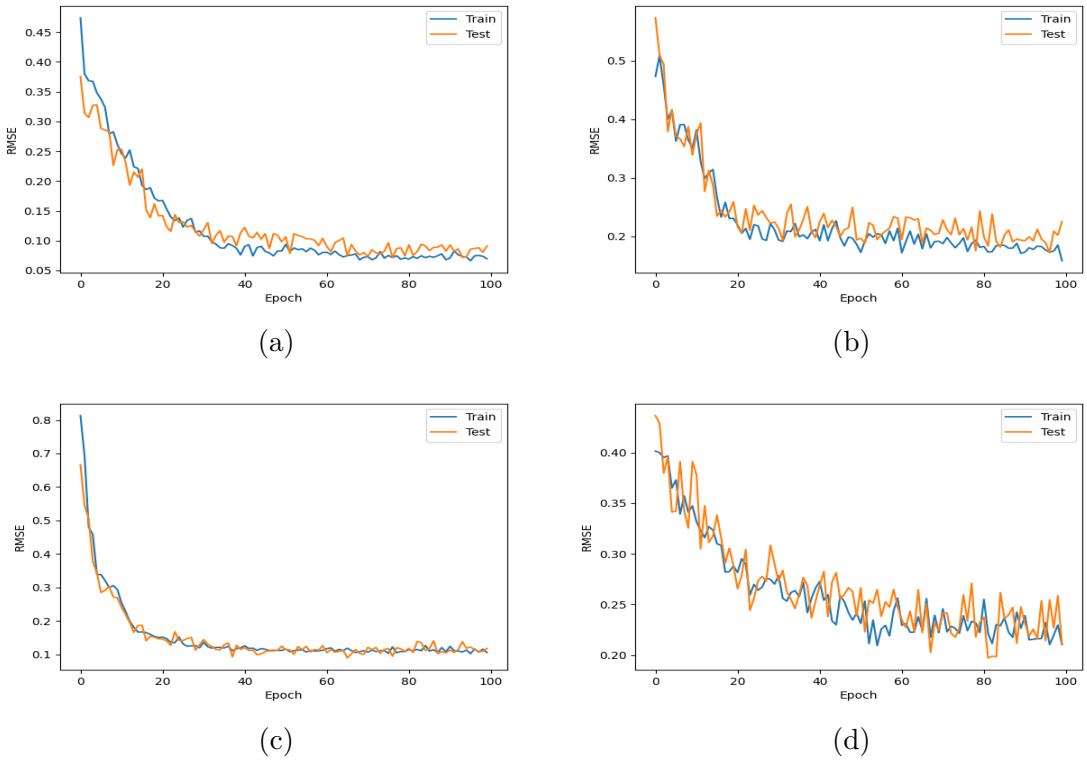


Figure 4.8 RMSE metric data from *sine* and *summed cosine waves* functions for the QMAC-based stochastic and 1-shot quantum LSTM models. (a) *sine* on QMAC-based stochastic model, (b) *sine* on 1-shot quantum model, (c) *summed cosine waves* on QMAC-based stochastic model, (d) *summed cosine waves* on 1-shot quantum model.

was devised to generate an exact expectation on the stochastic models, we can expect that they would reduce to the classical model. Therefore, the multi-shot stochastic models presented in this section use a standard of 100 samples to experimentally generate their expectation values.

Aiming to form a stochastic model that is as close to the quantum version as possible, this multi-shot inference treatment has been applied to the QMAC LSTM model. While a quantum model must completely rerun its circuits to construct an experimental expectation value, classical computing grants the ability to perform the standard MAC operation a single time and repeatedly quantize the result to accumulate the desired expectation. This modification to the previously established

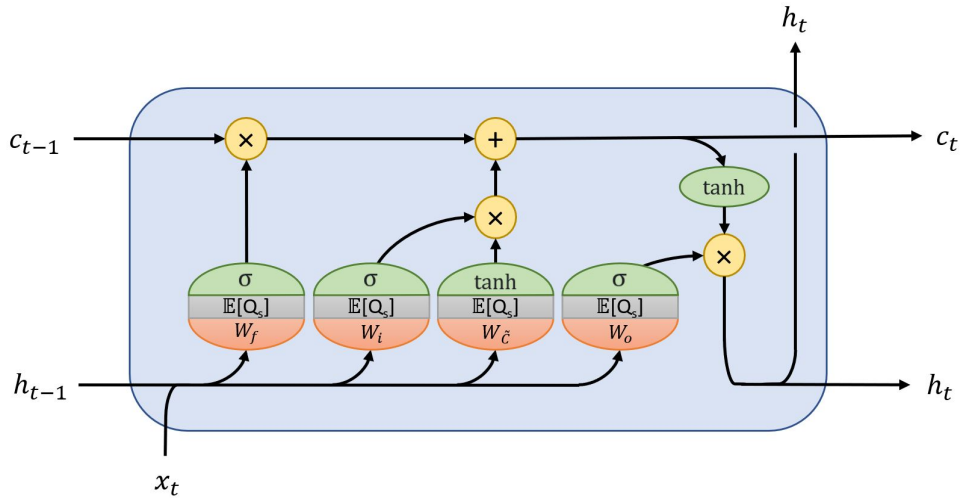


Figure 4.9 Graphical representation of an expectation valued QMAC-based LSTM cell.

QMAC LSTM cell architecture is illustrated by Figure 4.9. Results gathered using this model have been tabulated into Table 4.9.

Table 4.9 Metrics from the best epoch for the QMAC-based stochastic model with MAC-based multi-shot inference over all experiments.

MAC-Based Multi-Shot Quantized MAC LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0031	0.0430	0.9937	0.0043	0.0522	0.9913
Sawtooth	0.0409	0.0500	0.8618	0.0783	0.0846	0.7833
Summed Waves	0.0083	0.0720	0.9669	0.0088	0.0735	0.9644
Oscillator	0.0026	0.0382	0.9861	0.0018	0.0342	0.9414

This method proves to be quite successful; for all experiments, the performance has been improved from the normal QMAC LSTM model. Surprisingly, it even achieved a better RMSE value than both the classical and quantum models on the *sawtooth* experiment. These results are enough to confirm the potential possessed by stochastic models for producing greater results through experimentally computing expectation values for their non-deterministic features.

With the QMAC-based model benefiting from the boon of a multi-shot evaluation,

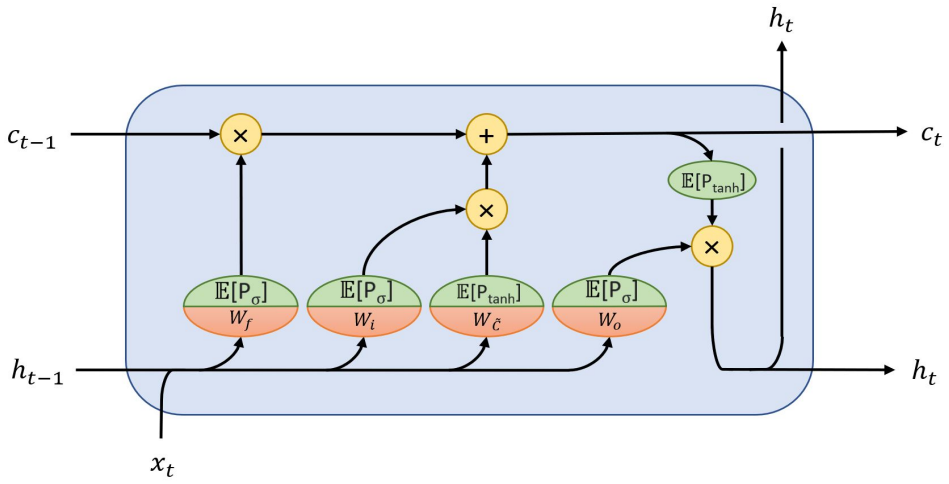


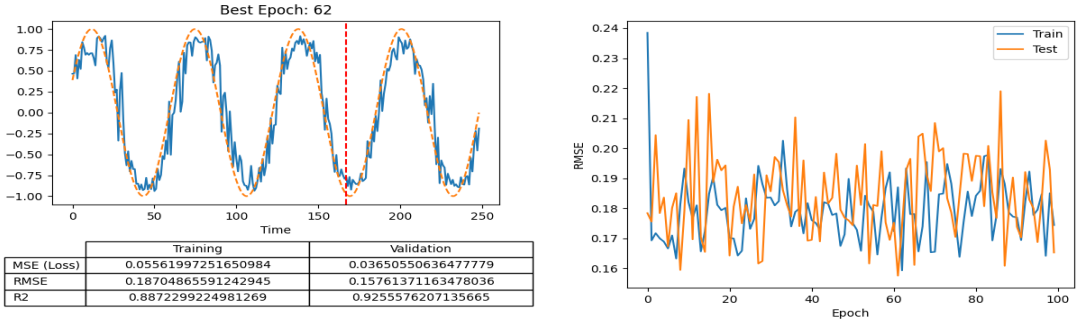
Figure 4.10 Graphical representation of an expectation valued BSN-based LSTM cell.

there seemed to be a prime opportunity to revisit the failed BSN-based stochastic model. As with the multi-shot QMAC approach, the MAC operations need not be repeated to recurrently apply the probabilistic activation functions. Once calculated, expectation value over BSN result is passed to the subsequent processes as the output from any normal neuron would be. The LSTM cell architecture using this technique is shown in Figure 4.10, and the results gathered are listed in Table 4.10.

Table 4.10 Metrics from the best epoch for the BSN-based stochastic model with neuron-based multi-shot inference over all experiments.

Neuron-Based Multi-Shot Binary Stochastic Neuron LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.056	0.1870	0.8872	0.0365	0.1576	0.9256
Sawtooth	0.1086	0.2074	0.6326	0.1219	0.2203	0.6627
Summed Waves	0.1712	0.3410	0.3160	0.1217	0.2787	0.5066
Oscillator	0.0823	0.2351	0.5589	0.0246	0.1183	0.2062

While still far from the performances achieved by the in-house, quantum, or QMAC models, computation of an expectation value over the BSNs has vastly improved the results exhibited in Table 4.6. Giving this model version's performance



(a) (b)

Figure 4.11 Sample performance for the BSN-based stochastic LSTM, using neuron-based multi-shot inference, on *sine* function. (a) best epoch data, (b) RMSE metric data.

on *sine* as an example in Figure 4.11, inference still greatly fluctuates between iterations rather than effectively converging to an optimum. Even so, this formulation is capable of achieving reasonable results. It even managed to slightly outperform the quantum model with 1-shot evaluations.

Clearly this technique is a powerful method for working with stochastic models. By averaging together results, they are more effectively able to converge towards the patterns that may be obscured by their own stochasticity. As a final reconfiguration for this type of system, both the QMAC and BSN LSTM models have been reworked to produce experimental expectation values on their final recurrent cell output. In this setup, the LSTM cell internals function as indicated in Section 4.4 and an expectation value is taken on the value fed from the final recurrent iteration to the fully-connected layer. As such, the LSTM cells function like a stochastic unit as a whole. Results for both the BSN and QMAC LSTMs are given in Tables 4.11 and 4.12.

As expected, both sets of results indicate that the approach is effective at improving stochastic learning. While performing to the same relative degree as the other multi-shot inference models presented in this section, they manage to refine the outcome produced by the initial stochastic models. There isn't much to note about the

Table 4.11 Metrics from the best epoch for the QMAC-based stochastic model with cell-based multi-shot inference over all experiments.

Cell-Based Multi-Shot Quantized MAC LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0040	0.0509	0.9918	0.0053	0.0612	0.9892
Sawtooth	0.0418	0.0623	0.8587	0.0782	0.0924	0.7837
Summed Waves	0.0181	0.1028	0.9275	0.0204	0.1066	0.9175
Oscillator	0.0119	0.0759	0.9363	0.0024	0.0340	0.9224

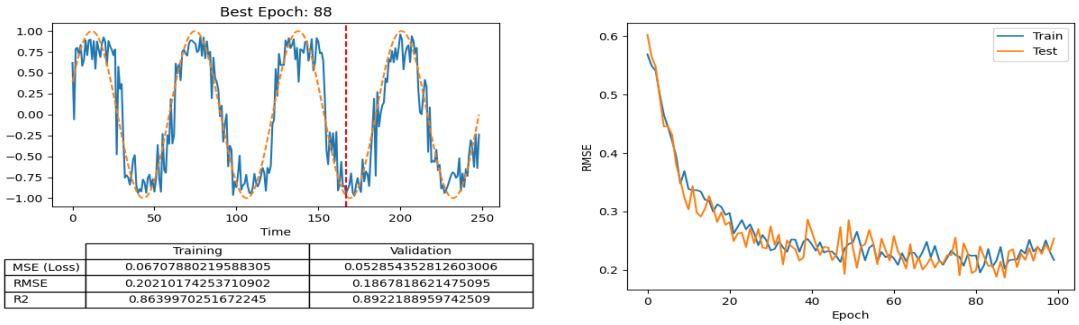
Table 4.12 Metrics from the best epoch for the BSN-based stochastic model with cell-based multi-shot inference over all experiments.

Cell-Based Multi-Shot Binary Stochastic Neuron LSTM Results						
Experiments	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
Sine	0.0671	0.2021	0.8640	0.0529	0.1868	0.8922
Sawtooth	0.1431	0.2801	0.5161	0.1378	0.2679	0.6186
Summed Waves	0.1678	0.3324	0.3296	0.1189	0.2723	0.5179
Oscillator	0.1334	0.3103	0.2850	0.0288	0.1382	0.0734

cell-based multi-shot QMAC LSTM that differs from its MAC-based counterpart, but it would seem that the cell-based multi-shot BSN LSTM experiences a more gradual convergence towards its optimum than the prior two versions. With the *sine* experiment again being presented as the example in Figure 4.12, the performance fluctuations are still exhibited but there is more of a trend present than previously seen.

4.7 Final Analysis of Results

Having now completed the analysis of each separate variety of the model studied, this work begins to wrap up by considering all results reported. Every previous metric result table has been consolidated into Tables 4.13-4.16 by their individual experiments for ease of viewing. Here the shorthand "multi-neuron", "multi-MAC",



(a) (b)

Figure 4.12 Sample performance for the BSN-based stochastic LSTM, using cell-based multi-shot inference, on *sine* function. (a) best epoch data, (b) RMSE metric data.

and "multi-cell" is adopted to reference multi-shot inference by taking an expectation value on binary stochastic neurons, quantized MAC output, and final LSTM cell result, respectively.

Table 4.13 Best performance metrics for all models on *sine* experiments.

Sine						
Model	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
In-house	0.0015	0.0315	0.9969	0.0032	0.0500	0.9933
Quantum	0.0028	0.0408	0.9944	0.0042	0.0554	0.9914
1-Shot Quantum	0.0510	0.1729	0.8965	0.0516	0.1718	0.8947
DR Quantized	0.0090	0.0735	0.9818	0.0104	0.0823	0.9788
SR Quantized	0.0108	0.0811	0.9781	0.0049	0.0529	0.9900
BSN	0.4735	0.6043	0.0400	0.4358	0.5763	0.1113
Multi-Neuron BSN	0.056	0.1870	0.8872	0.0365	0.1576	0.9256
Multi-Cell BSN	0.0671	0.2021	0.8640	0.0529	0.1868	0.8922
QMAC	0.0084	0.0717	0.9830	0.0079	0.0715	0.9839
Multi-MAC QMAC	0.0031	0.0430	0.9937	0.0043	0.0522	0.9913
Multi-Cell QMAC	0.0040	0.0509	0.9918	0.0053	0.0612	0.9892

The first thing to notice is that the in-house classical and analytical quantum models perform nearly on par for the experiments tested. The former of these produces slightly better results for the simple periodic functions, while outclassed by the lat-

Table 4.14 Best performance metrics for all models on *sawtooth* experiments.

Sawtooth						
Model	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
In-house	0.0416	0.0484	0.8593	0.0769	0.0905	0.7870
Quantum	0.0407	0.0465	0.8624	0.0727	0.0907	0.7989
1-Shot Quantum	0.1146	0.2350	0.6124	0.1430	0.2603	0.6041
DR Quantized	0.0498	0.0655	0.8316	0.0701	0.1173	0.8060
SR Quantized	0.0512	0.1061	0.8269	0.0823	0.0972	0.7721
BSN	0.2806	0.4410	0.0508	0.3736	0.5147	-0.0342
Multi-Neuron BSN	0.1086	0.2074	0.6326	0.1219	0.2203	0.6627
Multi-Cell BSN	0.1431	0.2801	0.5161	0.1378	0.2679	0.6186
QMAC	0.0458	0.0819	0.8450	0.0650	0.1029	0.8200
Multi-MAC QMAC	0.0409	0.0500	0.8618	0.0783	0.0846	0.7833
Multi-Cell QMAC	0.0418	0.0623	0.8587	0.0782	0.0924	0.7837

Table 4.15 Best performance metrics for all models on *summed cosine wave* experiments.

Summed Waves						
Model	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
In-house	0.0051	0.0564	0.9797	0.0059	0.0625	0.9761
Quantum	0.0021	0.0318	0.9917	0.0045	0.0559	0.9817
1-Shot Quantum	0.0734	0.2217	0.7067	0.0635	0.1973	0.7425
DR Quantized	0.0227	0.1160	0.9092	0.0196	0.1104	0.9206
SR Quantized	0.0232	0.1209	0.9072	0.0130	0.0912	0.9472
BSN	0.2597	0.4133	-0.0373	0.2530	0.4025	-0.0253
Multi-Neuron BSN	0.1712	0.3410	0.3160	0.1217	0.2787	0.5066
Multi-Cell BSN	0.1678	0.3324	0.3296	0.1189	0.2723	0.5179
QMAC	0.0134	0.1184	0.9119	0.0134	0.0897	0.9455
Multi-MAC QMAC	0.0083	0.0720	0.9669	0.0088	0.0735	0.9644
Multi-Cell QMAC	0.0181	0.1028	0.9275	0.0204	0.1066	0.9175

ter on physical dynamics based experiments. Furthermore, as mentioned previously and clearly displayed in Appendix B, the quantum model converges to the optimal solutions more rapidly than the classical model. Although all trials used 100 training epochs as a standard, this means that realistic applications for the model version would need to learn over less iterations to approach its best possible performances.

Table 4.16 Best performance metrics for all models on *damped harmonic oscillator* experiments.

Damped Harmonic Oscillator						
Model	Training			Validation		
	MSE	RMSE	R^2	MSE	RMSE	R^2
In-house	0.0004	0.0140	0.9978	0.0005	0.0199	0.9842
Quantum	0.0008	0.0120	0.9959	3.2e-5	0.0049	0.9990
1-Shot Quantum	0.0338	0.1463	0.8186	0.0122	0.0873	0.6081
DR Quantized	0.0033	0.0434	0.9825	0.0010	0.0266	0.9690
SR Quantized	0.0060	0.0585	0.9679	0.0008	0.0199	0.9745
BSN	0.1830	0.3673	0.0192	0.0293	0.1457	0.0573
Multi-Neuron BSN	0.0823	0.2351	0.5589	0.0246	0.1183	0.2062
Multi-Cell BSN	0.1334	0.3103	0.2850	0.0288	0.1382	0.0734
QMAC	0.0249	0.1248	0.8665	0.0068	0.0642	0.7810
Multi-MAC QMAC	0.0026	0.0382	0.9861	0.0018	0.0342	0.9414
Multi-Cell QMAC	0.0119	0.0759	0.9363	0.0024	0.0340	0.9224

Regardless, both showed high levels of success for every test they were applied to. This should be logical; these models deterministically follow a specific set of rules and equations.

The story is different when the stochastic behavior of the quantum model is fully enabled. By evaluating its circuits with a single measurement sample rather than analytically computing the exact expectation, this model's performance significantly drops. The best validation RMSE values it manages to achieve is an order of magnitude higher for all trials than what is attained through use of exact expectations. While the results are still acceptable, this configuration is outperformed by the in-house classical model in all experiments analyzed. Although it still generally converges quicker than the classic model, the performance is far less stable across training iterations than its analytical version.

As an additional point, the VQC output in the 1-shot expectation case will be quantized since measurements only have a single value to average. Although the rotational angles that serve as the analogue for trainable weights are real valued, this

makes the version more comparable to the weight quantized models than its analytical implementation. In fact, these achieve better performance than the 1-shot quantum model in every case. Some of the weight quantized trials even reached performance ratings close to these initial deterministic models. As previously mentioned, however, the primary difference from the classical implementation for these is the quantized weights; it should be clear by this point, but this design does not modify the standard model in a manner that bring it closer to the function of its quantum version. Furthermore, with stochasticity present only in the training phase and when the quantization function is stochastic rounding, this class of model contributes little to the discussion on QML's stochastic nature.

By contrast, the BSN and QMAC models retain inherent stochasticity even during inference and are thus more relevant for this study. It should be noted that none of the stochastic versions as presented here managed to outperform the standard classical or quantum models. The exception to this was the QMAC LSTM model with MAC-based multi-shot inference on the *sawtooth* experiment, with an RMSE value of 0.0846; however, this can likely be explained through slight variations caused by its stochastic behavior. This conjecture is supported by the fact that this trial's training metrics are still surpassed by that of the in-house classical and analytical quantum models. Additionally, this was the experiment resulting in the lowest performance for all model versions examined; while the loss was to an acceptable degree, it seems that even the successful models struggled to learn the non-continuous feature of this function. Nonetheless, this is a very promising outcome for capabilities that stochastic methods can afford to classic ML.

Considering for a moment only the BSN implementations, its obvious that their standard use in LSTM models is a poor fit. As pointed out in Section 4.4, the R^2 scores for this model are so low that some are actually negative; this indicates that the model exceptionally fails to explain the variance present in the data it is

shown. With the advent of multi-shot expectation values, however, this problem is reasonably rectified; despite still falling short of the other implementations, the reconfiguration in both forms manages to account for a greater amount of variance in the data and as such produces much better R^2 scores. Given that BSNs use a discrete probability distribution for their output, this is logically expected. Similar to how the quantum model's expectation values will experimentally converge to its analytical solution with enough iterations, the expectations for BSNs using Equations 3.5 and 3.6 will converge towards the standard sigmoid and hyperbolic tangent functions, respectively. Indeed, if an exact expectation for these was analytically calculated then the result would be those exact functions, and the model would reduce to the standard classical implementation.

Regardless, the QMAC models outclass the BSN variants even with the use of expectation values. These stochastic implementations prove to be highly successful, with performances that approach the level achieved by the classical and quantum models. More importantly, however, this model version demonstrates the ability to outperform the quantum model using 1-shot evaluations, even without the application of expectation values. Its convergence speed also appears to be of a similar order, in contrast to what was exhibited by the analytical models described above.

Its worth briefly discussing here a facet of training ML (and QML) models that is often missing from discussions: the total runtime for the programs performing model training. Although this is an aspect that is dependent upon hardware specifications, an understanding of it can impart insight into other cost factors of training ML models such as power consumption and device resource strain. The runtime taken by each model to produce the results analyzed has been collected into Table 4.17. To reiterate, all of these times resulted from 100 training epochs each.

The shortest times are exhibited by the in-house, weight quantized, and single shot QMAC models. While all are on the order of a single minute, those using

Table 4.17 Runtime for all models across every experiment.

Model Runtimes				
Model	Sine	Sawtooth	Summed Waves	Oscillator
In-house	24.88 sec	25.57 sec	24.57 sec	23.91 sec
Quantum	56.34 min	57.13 min	55.56 min	52.82 min
1-Shot Quantum	4.816 hr	4.563 hr	4.530 hr	4.356 hr
DR Quantized	35.15 sec	35.83 sec	36.11 sec	36.08 sec
SR Quantized	40.76 sec	41.47 sec	42.12 sec	42.03 sec
BSN	2.529 min	2.508 min	2.475 min	2.565 min
Multi-Neuron BSN	3.491 hr	3.385 hr	3.471 hr	3.495 hr
Multi-Cell BSN	3.414 hr	3.389 hr	3.414 hr	3.640 hr
QMAC	48.38 sec	49.66 sec	49.52 sec	48.82 sec
Multi-MAC QMAC	30.97 min	28.21 min	30.52 min	28.42 min
Multi-Cell QMAC	41.69 min	43.85 min	43.48 min	42.03 min

stochastic rounding are slightly higher due to the generation of a random number as explained in Section 3.3. The single shot BSN model falls shortly behind this, with every neuron requiring random operations. These outcomes are the result of pseudo-random actions being hard for otherwise deterministic machines, and should be expected. The QMAC and BSN model versions using multi-shot expectations compound the additional runtime incurred by pseudo-random number generation. Although this is a minor detriment to the QMAC-based instances, driving runtimes to the order of about half an hour, the BSN models suffered more greatly, reaching times on the order of three and a half hours. Clearly, the lower amount of time required for QMAC implementations to converge on optimal solutions is analogous to their better performance over BSN models.

Conversely to the rapid runtimes achieved by the classical models, the quantum realizations had much higher times to completion. Considering statements made in Chapter 1, this is logical; it is a well documented fact that quantum dynamics are hard to classically simulate, even with the limited amount of 4 qubits. Despite this, the quantum model utilizing analytical expectation values demonstrate acceptable times on the order of approximately an hour. The 1-shot expectation trials for this model

version, however, are a different story; taking the longest of any implementations to complete the training procedure, these require times on the order of about four and a half hours. This is explained by the circuits' generation of stochastic results, with pseudo-random processes once more driving potential runtimes even higher. The combination of quantum simulation and probabilistic procedures results in an undesirable time factor.

When faced with the performance possible through stochastic techniques in QMAC versions, this is nigh unreasonable. Granted, this particular matter is rectified through the use of real quantum hardware rather than simulations. It would be remiss in this case, however, not to also consider the time incurred by relying on devices available through the cloud. As direct access to such hardware should not be currently expected, the time spent queuing a quantum task, waiting for the job's turn, and ultimately returning the results to local machines for every individual call to a quantum circuit may equivalently serve to swell runtime of QML models run via cloud computing.

With this, we consider the analysis of this work complete and once more return to the initial question of the investigation: **Can we achieve similar results to variational QML by incorporating stochasticity inspired by its architectures into classical machine learning models?** When attempting to answer this question, it is important take into account the perspective elaborated on in Section 4.5; when simulating quantum computation, it is common to exploit the advent of analytical expectation values for measurement operations. Indeed, unless it is otherwise specified, we can expect that results reported of these models take advantage of this concept. The stochasticity created by measurements is effectively removed in this manner, other than the residual presence of a probability distribution. Yet, by applying stochastic techniques inspired by the paradigm such as presented in Section 4.6, we can achieve results on par to the analytical QLSTM model. While the QML

model investigated can be argued to be performing marginably better, true applications to quantum hardware are not capable of giving exact expectations such as the analytical model produced. Although real measurement operations will theoretically converge to the analytical result when given samples from enough circuit iterations, we can similarly converge the results of the stochastic model presented by taking the an expectation over a greater number of shots. While the results might differ for other QML implementations and VQA circuit designs, we therefore arrive at the logical conclusion for the work's case study: **It is possible to realize similar performance to variational QML models through applications of stochasticity to classical models.**

Chapter 5

Discussion

While the core question of this investigation may not progress the state-of-the-art for QML, it raises some important implications for research into the topic. First and foremost, the standard for testing new approaches and models should be testbedded using quantum simulators. This may seem somewhat contrived, given that the end goal would be deployment using real quantum hardware. In fact, this may even be inefficient if direct access to such devices is available since quantum computation is innately hard to simulate classically. Despite these downsides, however, doing so will enable an unmediated analysis of the theoretical capabilities for the system in question. This outlook comes from the unique ability to analytically produce exact expectation values of measurements that simulating such systems poses. Certainly if such technology is available, new designs should be experimentally verified; however, such simulations will allow insight into the potential limits it might achieve. Although simulating frameworks utilizing a large number of qubits may be infeasible, application of VQA techniques are generally on a scale such that this is possible.

As a corollary to this point, the training for QML models to be implemented on quantum hardware should be carried out by simulations of the same design. Approaching the practice with this aspect grants models the ability to learn their optimizable parameters without the need to experimentally converge their measurement values. They may then be used for inference with the ideal parameters, perhaps leading to slightly better performance than what would otherwise be achievable.

Not discounting the statements just made, it is worth noting that the utilization

of quantum computing may not expressly benefit the field of machine learning. True quantum devices will require many iterations to converge their stochastic results to the theoretical output of circuits. For ML models that typically require a substantial number of layers and training epochs to realize desirable results, the device workload can add up quickly with such a requirement. This reduces the paradigm's overall usefulness within the discipline, save for scientific curiosity. Even once the fabled day that quantum computers are commercially available and somewhat common arrives, the overhead this ramification imparts means that their use in common ML applications will be suboptimal.

In certain circumstances, however, the consequences of this may be mitigated. Firstly, if the employment of VQA circuits is limited to only select portions of hybrid models then the impact of iterative measurement convergence will be reduced. For example, if a model consisting of fifteen layers utilizes a VQA circuit for only a single layer, then the overhead the technique incurs will be minimal compared to a model consisting of only VQA layers. This method of classical ML augmentation may indeed prove useful. Second, the use of QML for research purposes is largely be unaffected by the aforementioned concerns. These implementations do not necessitate near immediate results, and as such can take long periods to return results with little drawback. The final case to mention would be in systems that are wholly quantum. While these are currently extremely uncommon, if they exist at all yet, they present the opportunity to postpone measurements until model inference occurs, drastically reducing the overall number of expectation values that must be converged.

When faced with these considerations, ML practitioners and engineers interested in gaining an advantage over wholly classical methods may find application of the stochastic techniques presented in this work of greater potential than the quantum realizations. While the analysis found that the in-house classical model outperformed its stochastic realizations, the datasets tested were all fairly simple patterns and con-

tained a large number of data points. This is not the case for many real applications of the discipline; data may be sparse and the underlying patterns can appear non-discrete. In these situations, stochasticity presents an avenue for models to better generalize their training data and more easily adapt to unseen samples. Utilization of the stochastically rounded quantized MAC method demonstrated in this work provides an accessible approach for this purpose without a substantial increase in runtime.

It is also worth contemplating the prospect of applying the concept of experimental expectation values introduced herein to probabilistic devices. While BSNs proved unsuccessful in standard form for the model of this case study, they have documented use cases that fully exhibit their potential. Furthermore, unlike the other stochastic techniques of this work they may be realized directly in hardware via magnetoresistive random-access memory (MRAM) units [64, 66, 46, 47]. The magnetic tunneling junctions (MTJ) of these devices provide an energy barrier between its high and low resistance states such that it may be used as nonvolatile memory to store information [65]. If this energy barrier within the MTJ is reduced, however, thermal noise and fluctuations can cause the MRAM to stochastically switch between its states. Electrical circuit constructions using only three transistors and one such MTJ allow for a sigmoidal distribution of output voltages that depend upon the input voltage; this culminates in a massive energy and area advantage for implementing BSNs over what is possible with conventional circuits [15, 14, 28]. By pairing this physical realization with experimental expectation values as described by this work, a high performance BSN-based model can be achieved. [12]

As a final point of discussion, the possibility for a difference in findings with a fully quantum system must be assessed. A factor intrinsic to modern NISQ-era devices that wasn't considered in this study is the existence of noise. Whether through a hybrid or non-hybrid model, the lack of completely fault-tolerant quantum error

correction establishes such noise as an additional form of stochasticity. Even given enough iterations to converge measurements and effectively eliminate them as a source of stochasticity, QML models implemented on current quantum hardware will still experience minor stochastic fluctuations. While the primary advantage these models exhibit can be considered the result of wholly quantum features, device noise may impart a further unexpected advantage to the discipline. Although overall performance can be negatively impacted, future work might find this stochasticity to aid generalization of quantum models in the manner previously described in this chapter.

Chapter 6

Conclusion

Through an empirical case study, this work approached matters of classical ML and QML to examine the affect of stochasticity on performance. Implementation of in-house models enabled an analysis of the effects that stochastic techniques can produce. Further, construction of a quantum model through the PennyLane library allowed a first-hand comparison of the potential that the paradigm can afford to the practice as a whole. While commonly regarded with a sense of mysticism due to its foundations, complexity in contrast to classical computing, and the difficulties of physical hardware realizations, investigations of this nature into the possibilities afforded by quantum computing are critical to understanding its ideal applications when the technology becomes commonly available. The research herein thus serves to highlight that the intrinsically possessed stochastic nature of the paradigm can be realized through classical techniques. As such, its power may be better suited for non-stochastic approaches.

Despite these findings, it would be remiss to consider this conclusion as an absolute statement. By no means does an analysis on LSTM models inherently transfer its results to all existing neural network architectures. The hypothesis that variational QML's advantage is not a result of its stochastic nature should be verified on other quantum models as a matter of further work on the subject. There may indeed be some cases where fully stochastic measurement operations lead to better performance than relying upon their analytical solutions. Furthermore, there could be some benefit in leveraging VQA circuit designs in a limited fashion to augment standard models

with their stochastic results.

As an additional point of prospective future work, methods of incorporating stochasticity into ML models discussed by this study should be evaluated to a greater extent. Rather than posing the investigation as a comparison to fundamentally different systems, the field of ML can stand to benefit from analyses on the potential furnished by stochastic approaches. Mentioned in Chapter 5, the trials of this study were relatively simple and consisted of a large number of data points; research in the area indicates that this diminishes the capacity for improvements through stochasticity. Supporting more real experiments with this methodology can provide a greater demonstration of the techniques' capabilities at improving classical ML performance.

Similarly, a more thorough exploration of the relatively novel approach of quantizing MAC output through stochastic rounding can yield higher insight into its advantages. For example, modifying the selected quantization bitwidth or the number of expectation value samples may lead to more optimal performance. Furthermore, this study quantized all data to standard integer values in $\mathbb{Z}$; by using a different quantization resolution to produce fixed-point values or subsets of $\mathbb{Z}$, QMAC model performance may yet be further improved. Considerations such as these are out of the scope for this work, but are interesting nonetheless.

With this, we once more consider the initial question of this investigation as it reaches its conclusion. **Can we achieve similar results to variational QML by incorporating stochasticity inspired by its architectures into classical machine learning models?** It should be clear through the efforts of this analysis that **it is possible to realize similar performance to variational QML models through applications of stochasticity to classical models.** It is this author's hope that the stated outcome should help elucidate the nature of quantum computing's place in the ML community in the face of alternative methods.

Bibliography

[1] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.

[2] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[3] Mohammad H. Amin et al. "Quantum Boltzmann Machine". In: *Phys. Rev. X* 8 (2 May 2018), p. 021050. DOI: 10.1103/PhysRevX.8.021050. URL: https://link.aps.org/doi/10.1103/PhysRevX.8.021050.

[4] Eric R. Anschuetz and Yudong Cao. *Realizing Quantum Boltzmann Machines Through Eigenstate Thermalization*. 2019. DOI: 10.48550/ARXIV.1903.01359. URL: https://arxiv.org/abs/1903.01359.

[5] Paul Benioff. "Quantum mechanical Hamiltonian models of Turing machines". In: *Journal of Statistical Physics* 29 (Nov. 1982), pp. 515–546. DOI: 10.1007/BF01342185.

[6] Charles H. Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*. Bangalore, India, 1984, pp. 175–179.

[7] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2020. arXiv: 1811.04968 [quant-ph].

[8] Daniel Bernstein. "Introduction to post-quantum cryptography". In: Springer, Jan. 2009, pp. 1–14. ISBN: 978-3-540-88701-0. DOI: 10.1007/978-3-540-88702-7_1.

[9] E. Bernstein and U. Vazirani. "Quantum complexity theory". In: *Proceedings of the 25th ACM Symposium on Theory of Computation* (1993), pp. 11–20.

[10] Kishor Bharti et al. *Noisy intermediate-scale quantum (NISQ) algorithms*. 2021. arXiv: 2101.08448 [quant-ph].

[11] Jacob Biamonte et al. “Quantum machine learning”. In: *Nature* 549.7671 (Sept. 2017), pp. 195–202. ISSN: 1476-4687. DOI: 10.1038/nature23474. URL: http://dx.doi.org/10.1038/nature23474.

[12] William A. Borders et al. “Integer factorization using stochastic magnetic tunnel junctions”. In: *Nature* 573.7774 (2019), pp. 390–393. DOI: 10.1038/s41586-019-1557-9.

[13] Duncan Buell. *Fundamentals of Cryptography: Introducing Mathematical and Algorithmic Foundations*. Jan. 2021, p. 177. ISBN: 978-3-030-73491-6. DOI: 10.1007/978-3-030-73492-3.

[14] Kerem Y Camsari, Brian M Sutton, and Supriyo Datta. “P-bits for probabilistic spin logic”. In: *Applied Physics Reviews* 6.1 (2019), p. 011305.

[15] Kerem Yunus Camsari, Sayeef Salahuddin, and Supriyo Datta. “Implementing p-bits With Embedded MTJ”. In: *IEEE Electron Device Letters* 38.12 (2017), pp. 1767–1770. DOI: 10.1109/LED.2017.2768321.

[16] M. Cerezo et al. *Variational Quantum Algorithms*. 2020. arXiv: 2012.09265 [quant-ph].

[17] Samuel Yen-Chi Chen, Shinjae Yoo, and Yao-Lung L. Fang. “Quantum Long Short-Term Memory”. In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 8622–8626. DOI: 10.1109/ICASSP43922.2022.9747369.

[18] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*. 2016. arXiv: 1511.00363 [cs.LG].

[19] Matthieu Courbariaux et al. “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1”. In: *arXiv preprint arXiv:1602.02830* (2016).

[20] Prasanna Date and Thomas Potok. “Adiabatic quantum linear regression”. In: *Scientific Reports* 11.1 (Nov. 2021). DOI: 10.1038/s41598-021-01445-6.

[21] Vedran Dunjko and Hans J. Briegel. *Machine learning & artificial intelligence in the quantum domain*. 2017. arXiv: 1709.02779 [quant-ph].

[22] Mohammed Elbtity et al. “An In-Memory Analog Computing Co-Processor for Energy-Efficient CNN Inference on Mobile Devices”. In: *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2021, pp. 188–193. DOI: 10.1109/ISVLSI51109.2021.00043.

[23] Richard P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6-7 (1982), pp. 467–488. DOI: 10.1007/bf02650179.

[24] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman and Co., 1990. ISBN: 0716710455.

[25] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: *Neural computation* 12 (Oct. 2000), pp. 2451–71. DOI: 10.1162/089976600300015015.

[26] Yunchao Gong et al. “Compressing deep convolutional networks using vector quantization”. In: *arXiv preprint arXiv:1412.6115* (2014).

[27] Lov K. Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the 28th annual ACM symposium on Theory of computing - STOC 96* (1996). DOI: 10.1145/237814.237866.

[28] Orchi Hassan, Supriyo Datta, and Kerem Y Camsari. “Quantitative evaluation of hardware binary stochastic neurons”. In: *Physical Review Applied* 15.6 (2021), p. 064046.

[29] Geoffrey Hinton and Tijmen Tieleman. *Rmsprop: Divide the gradient by a running average of its recent magnitude*. 2012.

[30] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Pearson Addison-Wesley, 2014.

[31] Itay Hubara et al. “Binarized neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 4107–4115.

[32] Itay Hubara et al. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations”. In: *J. Mach. Learn. Res.* 18.1 (Jan. 2017), pp. 6869–6898. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=3122009.3242044.

[33] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An Empirical Exploration of Recurrent Network Architectures”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 2342–2350.

[34] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An introduction to quantum computing*. Oxford University Press, 2010.

[35] Iordanis Kerenidis and Alessandro Luongo. "Classification of the MNIST data set with quantum slow feature analysis". In: *Phys. Rev. A* 101 (6 June 2020), p. 062327. DOI: 10.1103/PhysRevA.101.062327. URL: https://link.aps.org/doi/10.1103/PhysRevA.101.062327.

[36] Fengfu Li, Bo Zhang, and Bin Liu. "Ternary weight networks". In: *arXiv preprint arXiv:1605.04711* (2016).

[37] Hao Li et al. *Training Quantized Nets: A Deeper Understanding*. 2017. arXiv: 1706.02379 [cs.LG].

[38] Shuang Liang et al. "FP-BNN: Binarized neural network on FPGA". In: *Neurocomputing* 275 (2018), pp. 1072–1086. ISSN: 0925-2312.

[39] Andrea Mari, Thomas R. Bromley, and Nathan Killoran. "Estimating the gradient and higher-order derivatives on quantum hardware". In: *Physical Review A* 103.1 (Jan. 2021). DOI: 10.1103/physreva.103.012405.

[40] Jarrod R McClean et al. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. DOI: 10.1088/1367-2630/18/2/023023. URL: https://doi.org/10.1088/1367-2630/18/2/023023.

[41] K. Mitarai et al. "Quantum circuit learning". In: *Physical Review A* 98.3 (Sept. 2018). DOI: 10.1103/physreva.98.032309.

[42] T. Monz et al. "Realization of a scalable Shor algorithm". In: *Science* 351.6277 (Mar. 2016), pp. 1068–1070. ISSN: 1095-9203. DOI: 10.1126/science.aad9480. URL: http://dx.doi.org/10.1126/science.aad9480.

[43] B. Moons et al. "Minimum energy quantized neural networks". In: *2017 51st Asilomar Conference on Signals, Systems, and Computers*. Oct. 2017, pp. 1921–1925. DOI: 10.1109/ACSSC.2017.8335699.

[44] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. 10th Anniversary. Cambridge University Press, 2019.

[45] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[46] Vaibhav Ostwal et al. "A novel compound synapse using probabilistic spin-orbit-torque switching for MTJ based deep neural networks". In: *arXiv preprint arXiv:1910.00171* (2019).

[47] Hossein Pourmeidani et al. “Probabilistic Interpolation Recoder for Energy-Error-Product Efficient DBNs With p-Bit Devices”. In: *IEEE Transactions on Emerging Topics in Computing* 9.4 (2021), pp. 2146–2157. DOI: 10.1109/TETC.2020.2965079.

[48] Enrico Prati. “Quantum neuromorphic hardware for quantum artificial intelligence”. In: *Journal of Physics: Conference Series* (2017). DOI: 10.1088/1742-6596/880/1/012018.

[49] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[50] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[51] Maria Schuld and Nathan Killoran. “Quantum Machine Learning in Feature Hilbert Spaces”. In: *Phys. Rev. Lett.* 122 (4 Feb. 2019), p. 040504. DOI: 10.1103/PhysRevLett.122.040504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.040504.

[52] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. “The Quest for a Quantum Neural Network”. In: *Quantum Information Processing* 13.11 (Nov. 2014), pp. 2567–2586. ISSN: 1570-0755.

[53] Maria Schuld et al. “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3 (Mar. 2019). DOI: 10.1103/physreva.99.032331.

[54] Peter W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 124–134. DOI: 10.1109/sfcs.1994.365700.

[55] D. Simon. “On the power of quantum computation”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994). DOI: 10.1109/sfcs.1994.365701.

[56] Francesco Tacchino et al. “An artificial neuron implemented on an actual quantum processor”. In: *npj Quantum Information* 5.1 (2019). DOI: 10.1038/s41534-019-0140-4.

[57] Lieven M. K. Vandersypen et al. “Experimental realization of Shors quantum factoring algorithm using nuclear magnetic resonance”. In: *Nature* 414.6866 (2001), pp. 883–887. DOI: 10.1038/414883a.

[58] Guillaume Verdon et al. *Learning to learn with quantum neural networks via classical neural networks*. 2019. arXiv: 1907.05415 [quant-ph].

[59] John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018. DOI: 10.1017/9781316848142.

[60] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[61] Ronald J. Williams and Jing Peng. "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories". In: *Neural Computation* 2.4 (1990), pp. 490–501. DOI: 10.1162/neco.1990.2.4.490.

[62] A. Chi-Chih Yao. "Quantum circuit complexity". In: *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science* (1993). DOI: 10.1109/sfcs.1993.366852.

[63] Stathis Zachos. "Robustness of probabilistic computational complexity classes under definitional perturbations". In: *Information and Control* 54.3 (1982), pp. 143–154. DOI: 10.1016/s0019-9958(82)80019-3.

[64] R. Zand et al. "Low-Energy Deep Belief Networks Using Intrinsic Sigmoidal Spintronic-based Probabilistic Neurons". In: *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. GLSVLSI '18. Chicago, IL, USA: ACM, 2018, pp. 15–20. ISBN: 978-1-4503-5724-1. DOI: 10.1145/3194554.3194558.

[65] Ramtin Zand, Arman Roohi, and Ronald F DeMara. "Fundamentals, modeling, and application of magnetic tunnel junctions". In: *Nanoscale Devices*. CRC Press, 2018, pp. 337–368.

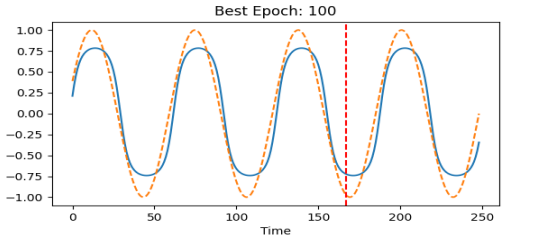
[66] Ramtin Zand et al. "Composable Probabilistic Inference Networks Using MRAM-based Stochastic Neurons". In: *J. Emerg. Technol. Comput. Syst.* 15.2 (Mar. 2019), 17:1–17:22. ISSN: 1550-4832. DOI: 10.1145/3304105.

[67] Shuchang Zhou et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[68] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. "Variational quantum Boltzmann machines". In: *Quantum Machine Intelligence* 3.1 (Feb. 2021). ISSN: 2524-4914. DOI: 10.1007/s42484-020-00033-7. URL: http://dx.doi.org/10.1007/s42484-020-00033-7.

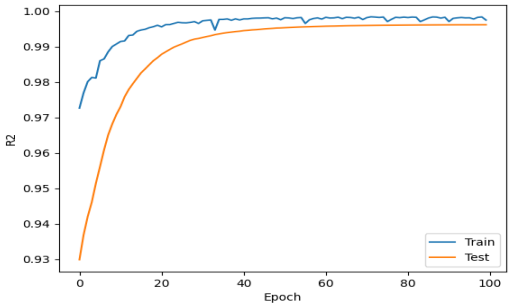
Appendix A

Classical Result Graphics

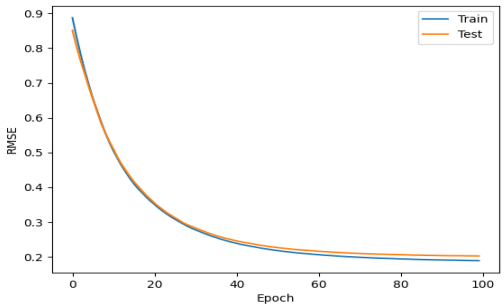


	Training	Validation
MSE (Loss)	0.035850875079631805	0.041174981743097305
RMSE	0.18934327363967896	0.2029161900281906
R2	0.9974665641784668	0.9961483478546143

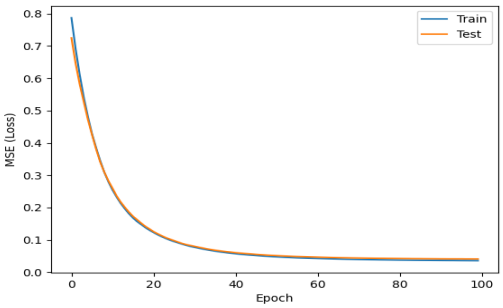
(a) final epoch prediction



(b) R^2 metric data

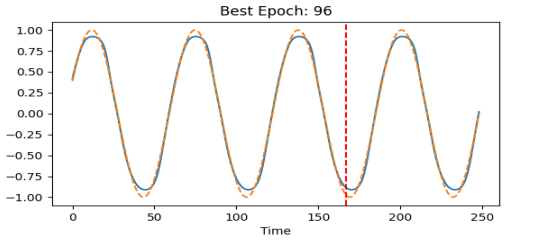


(c) RMSE metric data



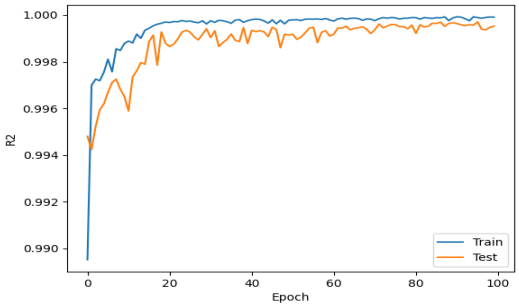
(d) MSE metric data

Figure A.1 Metrics and final epoch prediction of the TensorFlow LSTM, optimized by *standard gradient descent*, for *sine*.

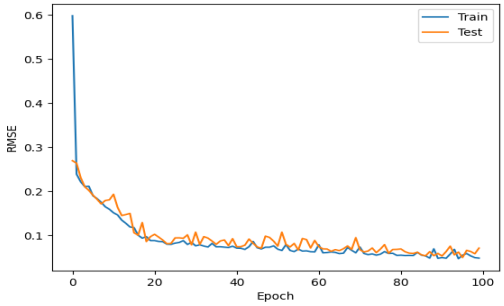


	Training	Validation
MSE (Loss)	0.0030444045551121235	0.0024955265689641237
RMSE	0.05517612397670746	0.04995524510741234
R2	0.9998880624771118	0.9997047781944275

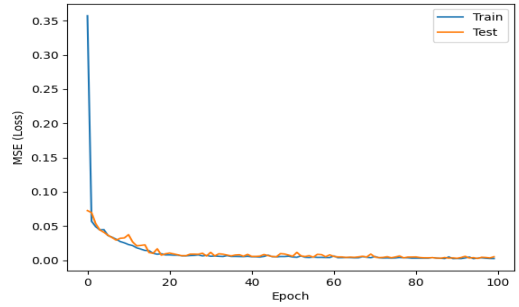
(a) final epoch prediction



(b) R^2 metric data

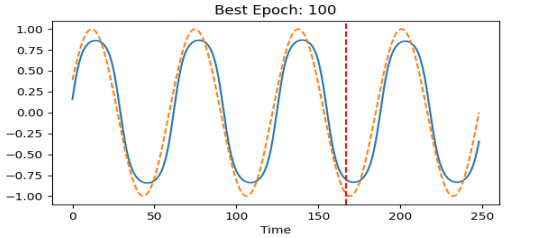


(c) RMSE metric data



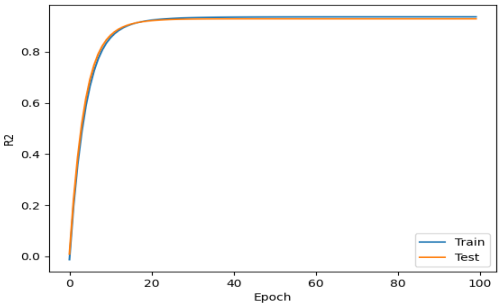
(d) MSE metric data

Figure A.2 Metrics and final epoch prediction of the TensorFlow LSTM, optimized by *RMSprop*, for *sine*.

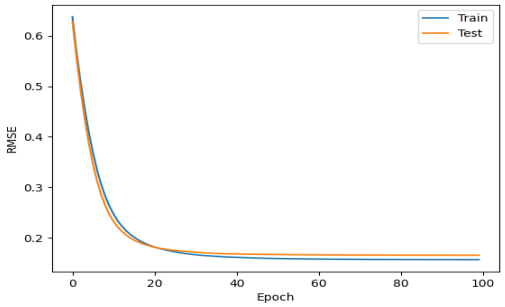


	Training	Validation
MSE (Loss)	0.030884286323533593	0.03458222576444237
RMSE	0.15673111554661306	0.16560281416701653
R2	0.9373817856299532	0.9294795929906735

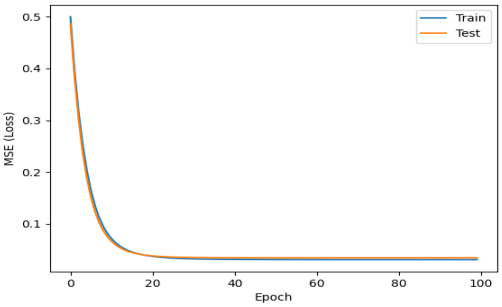
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

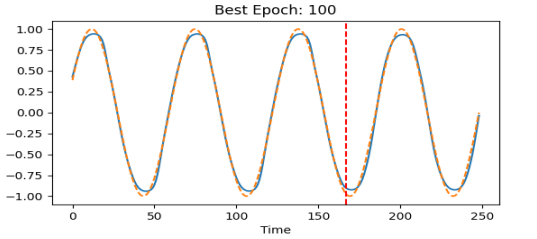


(c) RMSE metric data



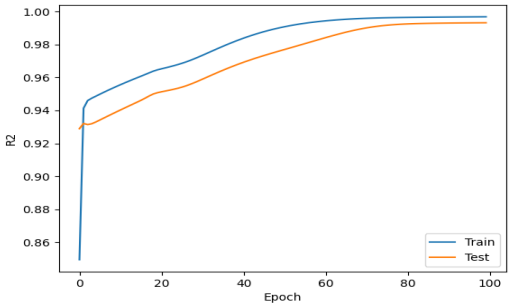
(d) MSE metric data

Figure A.3 Metrics and best epoch data of the in-house LSTM, optimized by *standard gradient descent*, for *sine*.

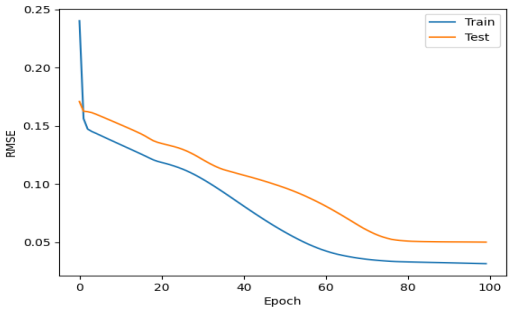


	Training	Validation
MSE (Loss)	0.0015187249160366519	0.003293368874945428
RMSE	0.03148512911492807	0.05001459463383491
R2	0.9969207693075605	0.9932841305509086

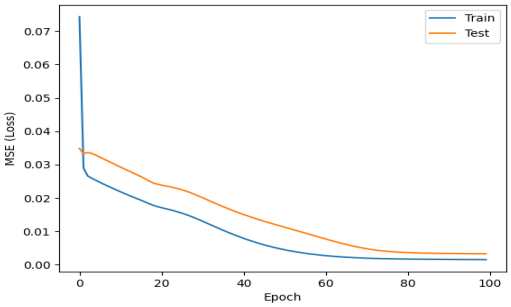
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

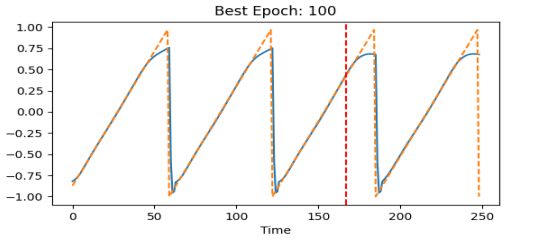


(c) RMSE metric data



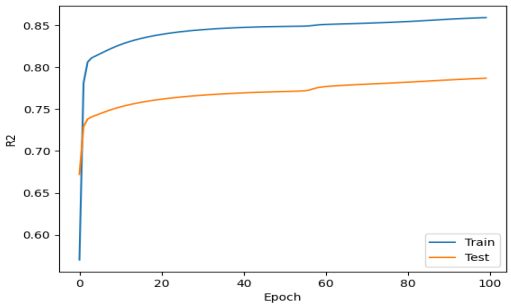
(d) MSE metric data

Figure A.4 Metrics and best epoch data of the in-house LSTM, optimized by *RMSprop*, for *sine*.

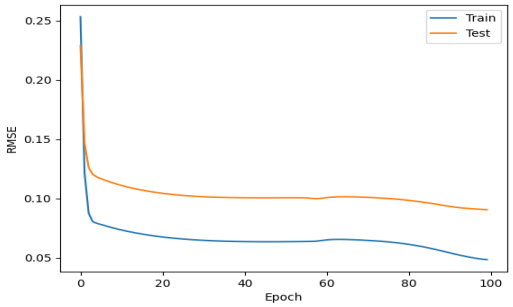


	Training	Validation
MSE (Loss)	0.04158891527790057	0.07693928916870575
RMSE	0.048435784765172525	0.09053723713262961
R2	0.8593429526115647	0.7870420186079365

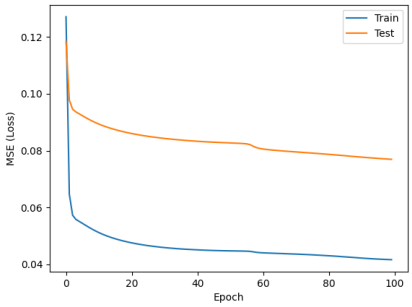
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

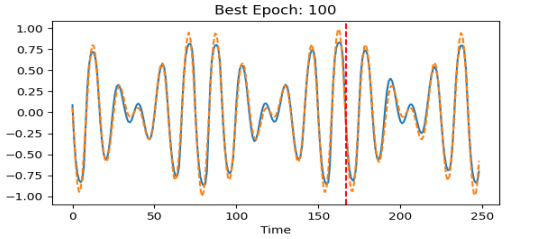


(c) RMSE metric data



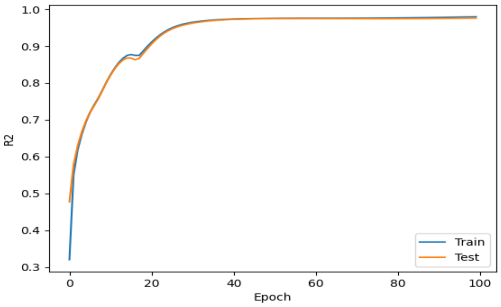
(d) MSE metric data

Figure A.5 Metrics and best epoch data of the in-house LSTM for *sawtooth*.

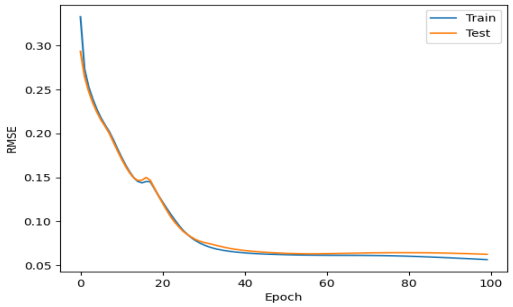


	Training	Validation
MSE (Loss)	0.005086806794660452	0.005896387095410197
RMSE	0.05642937632985515	0.06254839371019741
R2	0.9796790421272928	0.9761020555566849

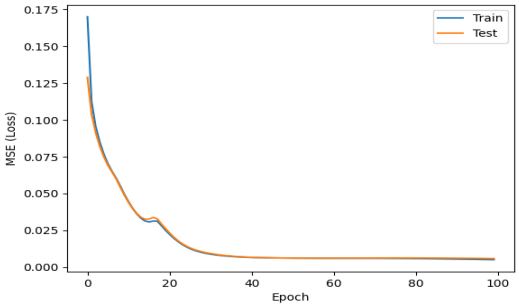
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

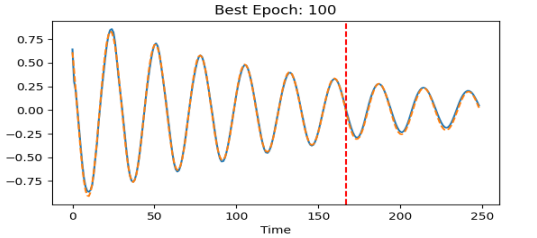


(c) RMSE metric data



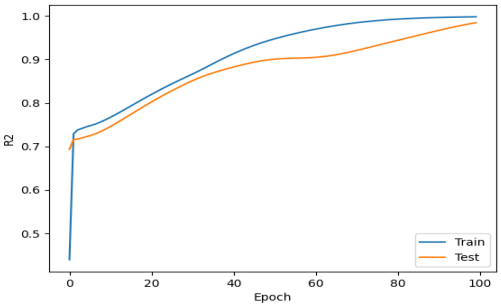
(d) MSE metric data

Figure A.6 Metrics and best epoch data of the in-house LSTM for *summed cosine waves*.

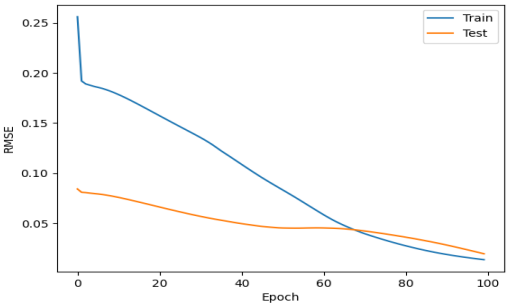


	Training	Validation
MSE (Loss)	0.0004154965476210655	0.0004913315637943508
RMSE	0.013982733549190933	0.019885483281059903
R2	0.9977731398217904	0.9841712947319554

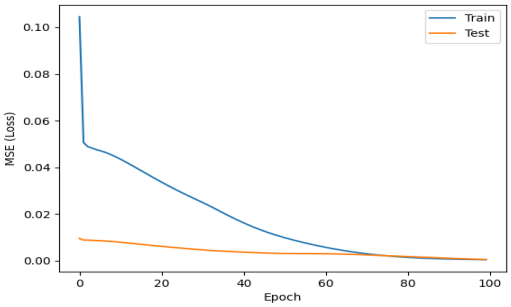
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data



(c) RMSE metric data

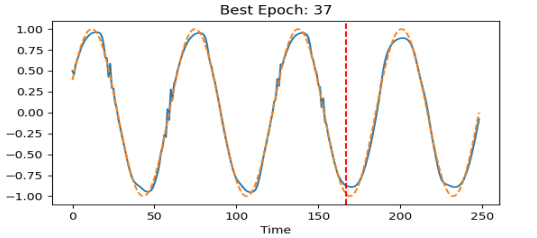


(d) MSE metric data

Figure A.7 Metrics and best epoch data of the in-house LSTM for *damped harmonic oscillator*.

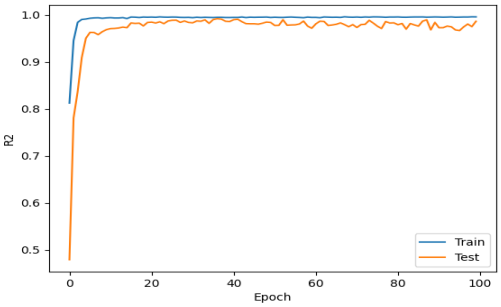
Appendix B

Quantum Result Graphics

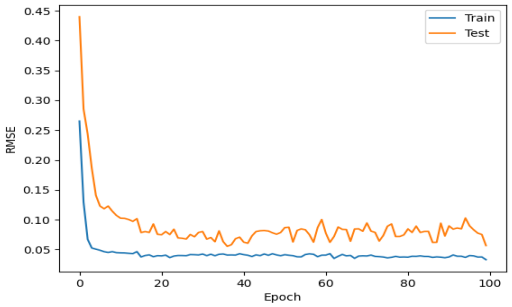


	Training	Validation
MSE (Loss)	0.002769238798994225	0.004195457172669873
RMSE	0.04079787534134215	0.05542619261576469
R2	0.9943853392411508	0.9914445828907251

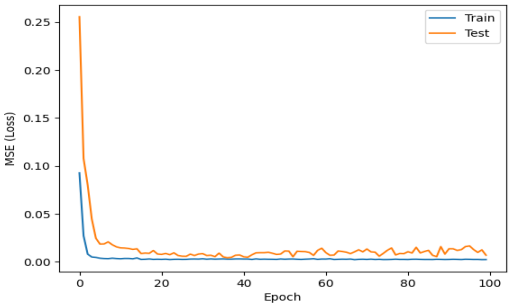
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

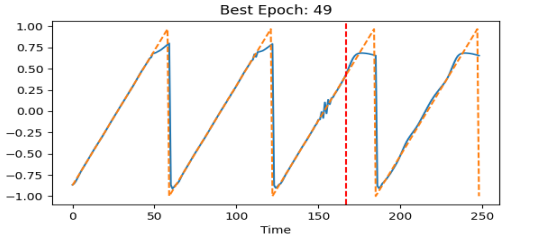


(c) RMSE metric data



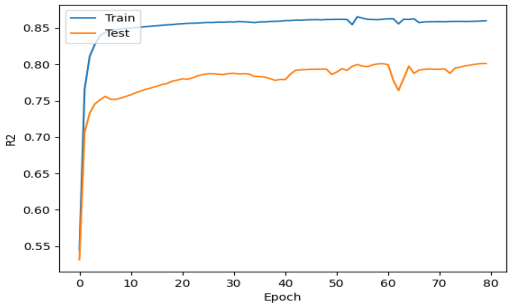
(d) MSE metric data

Figure B.1 Metrics and best epoch data of the quantum LSTM, utilizing the simulator, for *sine*.

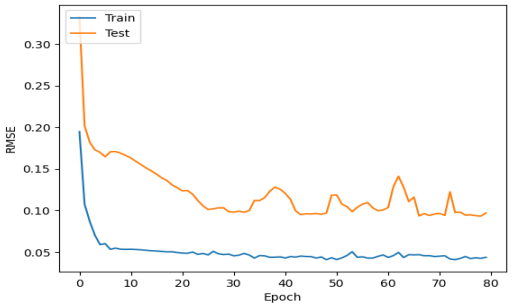


	Training	Validation
MSE (Loss)	0.04084948122961964	0.0745985306662459
RMSE	0.04104419467207718	0.09714465193360684
R2	0.8618437750677261	0.7935209426777129

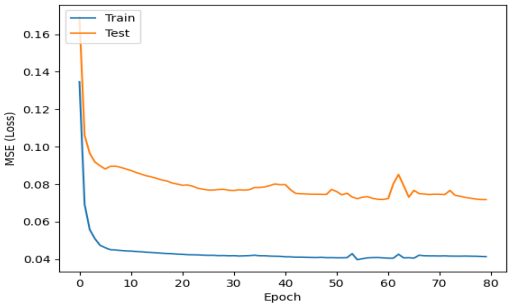
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

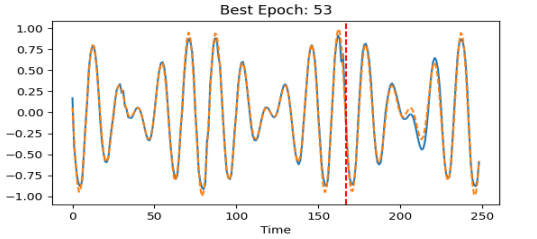


(c) RMSE metric data



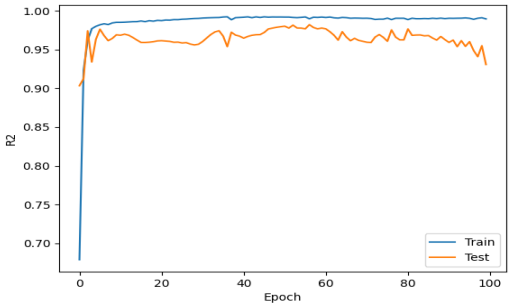
(d) MSE metric data

Figure B.2 Metrics and best epoch data of the quantum LSTM, utilizing the simulator, for *sawtooth*.

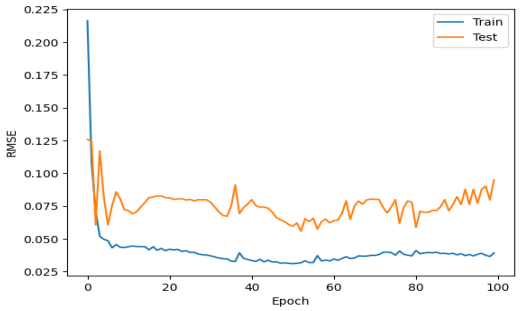


	Training	Validation
MSE (Loss)	0.0020891445701134077	0.004514959258799138
RMSE	0.03181489967426891	0.05591138003461332
R2	0.9916542105154249	0.9817009557378112

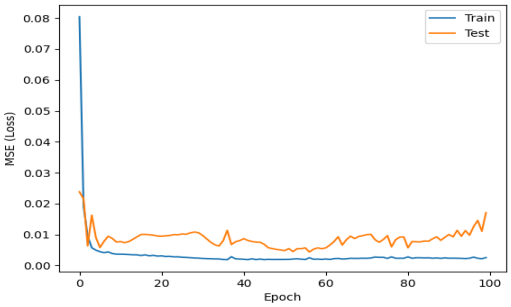
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

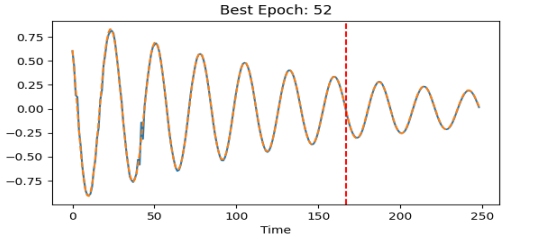


(c) RMSE metric data



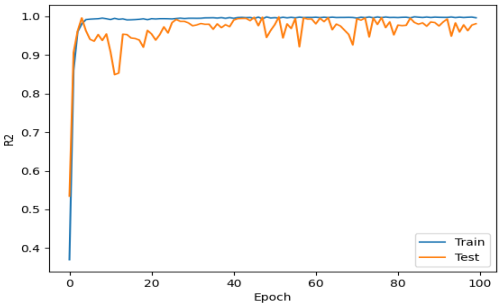
(d) MSE metric data

Figure B.3 Metrics and best epoch data of the quantum LSTM, utilizing the simulator, for *summed cosine waves*.

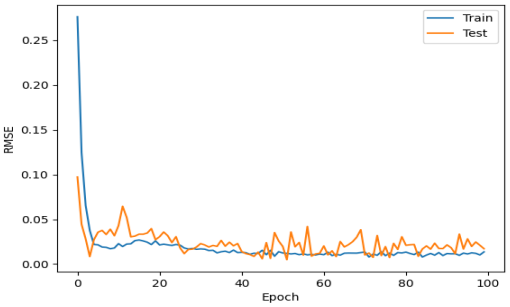


	Training	Validation
MSE (Loss)	0.0007682967434116345	3.2193574681392475e-05
RMSE	0.01197397228364036	0.004924033622605255
R2	0.995882301655208	0.9989628539423544

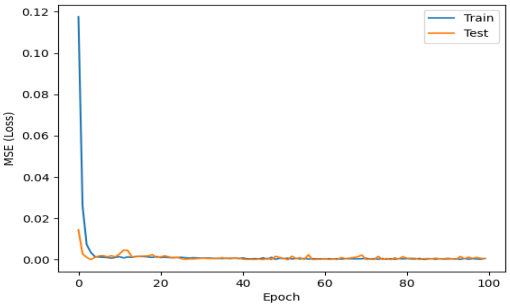
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

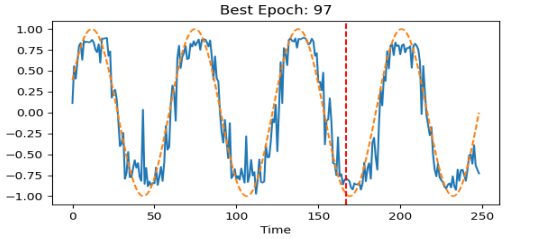


(c) RMSE metric data



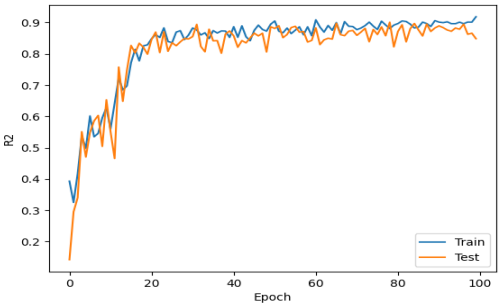
(d) MSE metric data

Figure B.4 Metrics and best epoch data of the quantum LSTM, utilizing the simulator, for *damped harmonic oscillator*.

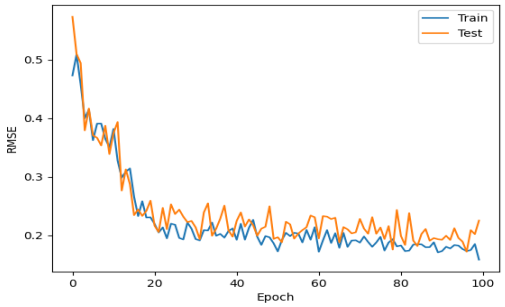


	Training	Validation
MSE (Loss)	0.05103855328978399	0.05161842413394331
RMSE	0.17289238717092806	0.1717942177531231
R2	0.896518795013445	0.8947392124024565

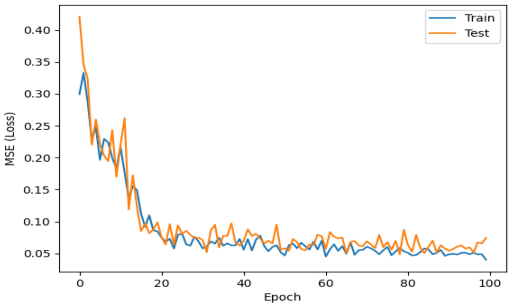
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

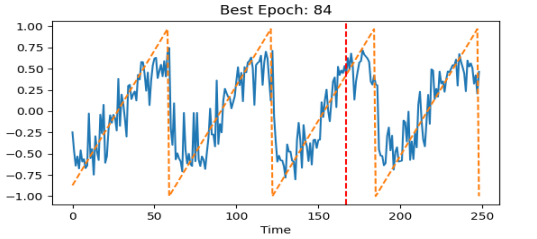


(c) RMSE metric data



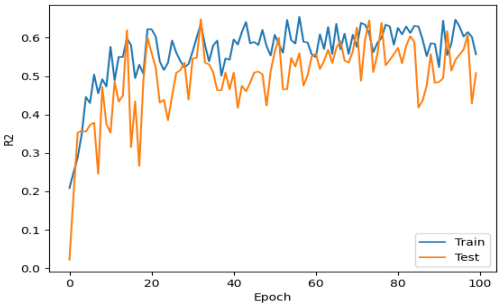
(d) MSE metric data

Figure B.5 Metrics and best epoch data of the quantum LSTM, utilizing the simulator with 1 shot expectation value, for *sine*.

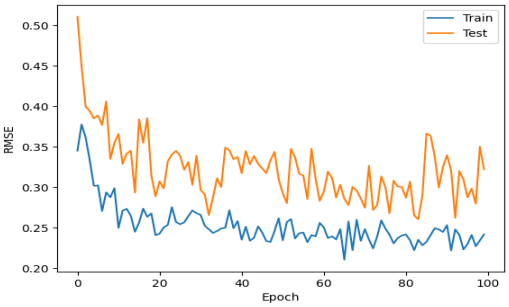


	Training	Validation
MSE (Loss)	0.11459760188638511	0.14304980482363597
RMSE	0.23498217862503357	0.26027443870363465
R2	0.6124217187900904	0.6040566863963357

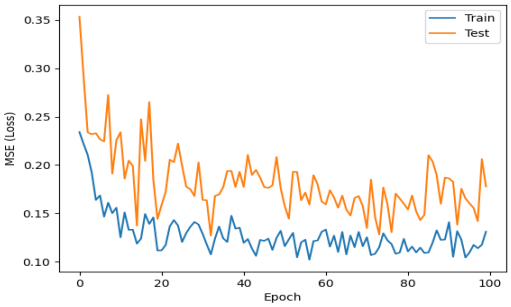
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

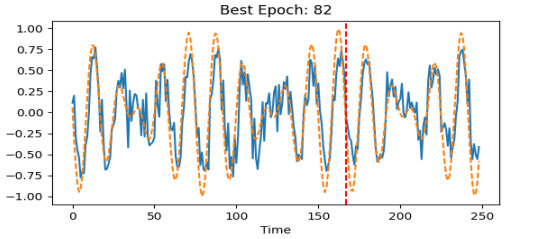


(c) RMSE metric data



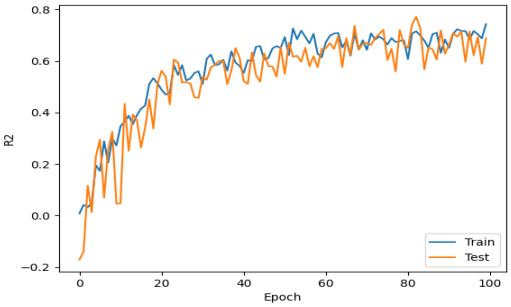
(d) MSE metric data

Figure B.6 Metrics and best epoch data of the quantum LSTM, utilizing the simulator with 1 shot expectation value, for *sawtooth*.

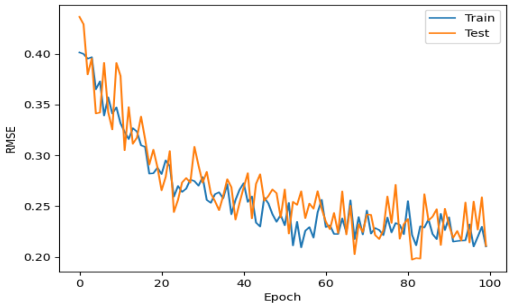


	Training	Validation
MSE (Loss)	0.0734270768078722	0.06354098556297992
RMSE	0.2216550768633564	0.1973089575318687
R2	0.7066708838442755	0.7424695966066565

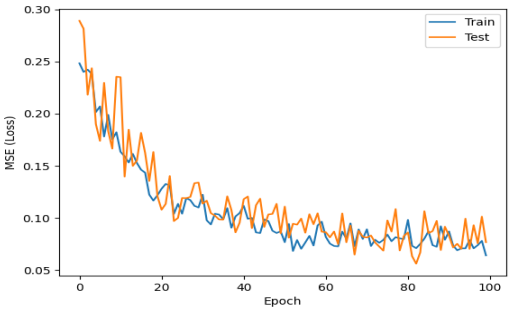
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

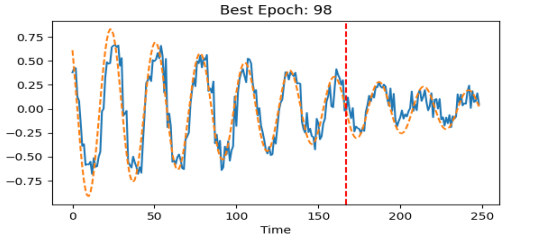


(c) RMSE metric data



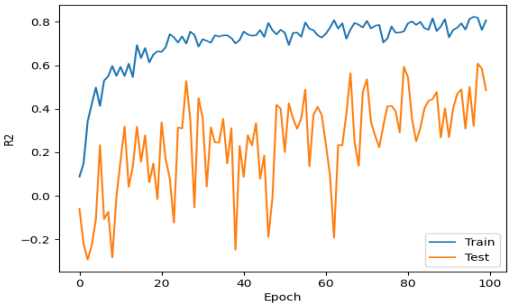
(d) MSE metric data

Figure B.7 Metrics and best epoch data of the quantum LSTM, utilizing the simulator with 1 shot expectation value, for *summed cosine waves*.

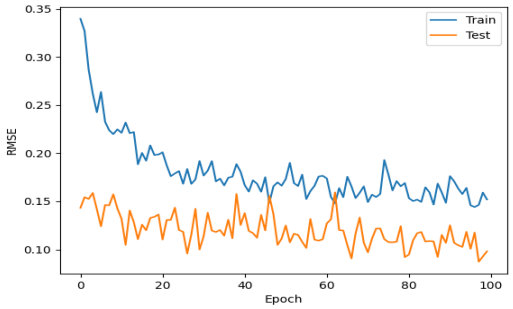


	Training	Validation
MSE (Loss)	0.03384297536328484	0.01216366017213055
RMSE	0.14626211646091508	0.0873140306941357
R2	0.8186180482721183	0.6081363206952222

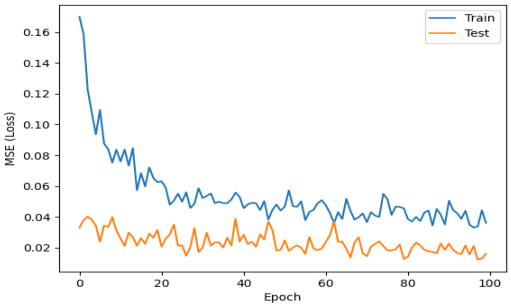
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data



(c) RMSE metric data

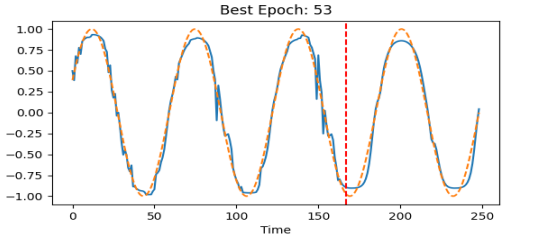


(d) MSE metric data

Figure B.8 Metrics and best epoch data of the quantum LSTM, utilizing the simulator with 1 shot expectation value, for *damped harmonic oscillator*.

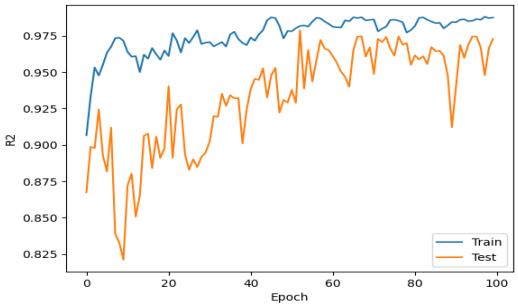
Appendix C

Quantized Result Graphics

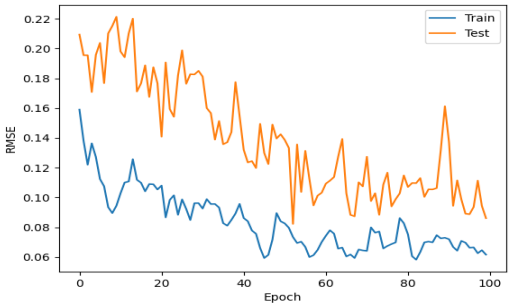


	Training	Validation
MSE (Loss)	0.00897416788679683	0.010405398417205896
RMSE	0.07347747094102537	0.0822796673354286
R2	0.9818047805074248	0.9787812115832621

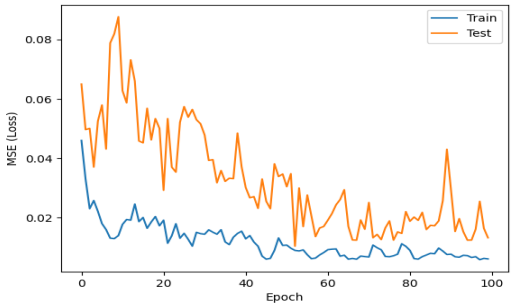
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

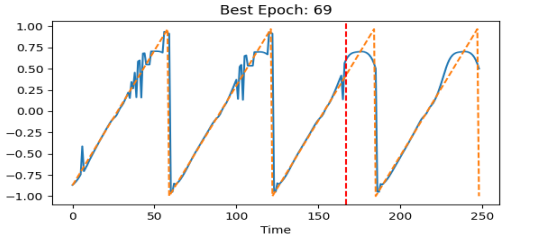


(c) RMSE metric data



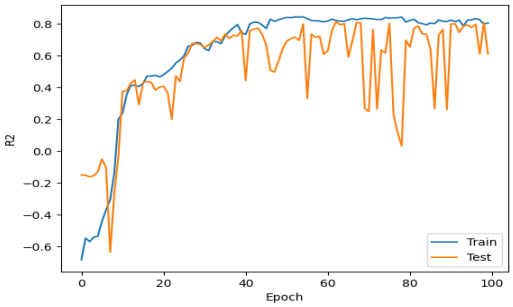
(d) MSE metric data

Figure C.1 Metrics and best epoch data of the quantized LSTM, utilizing deterministic rounding, for *sine*.

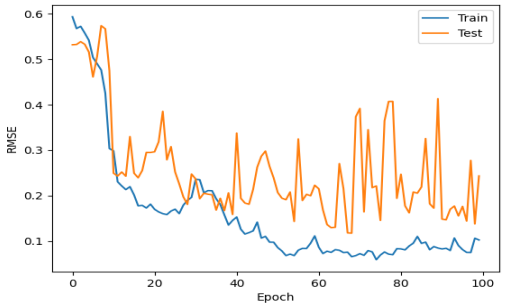


	Training	Validation
MSE (Loss)	0.04978400122233501	0.07010300365907474
RMSE	0.06549101771575115	0.11730793027259329
R2	0.8316265146055191	0.8059639709430899

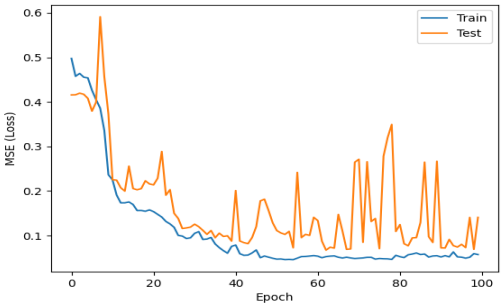
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

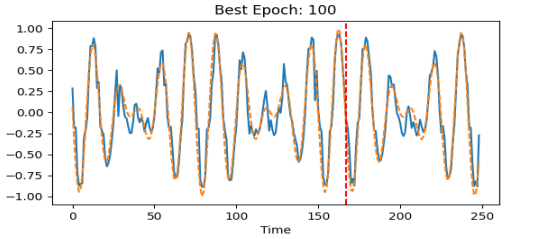


(c) RMSE metric data



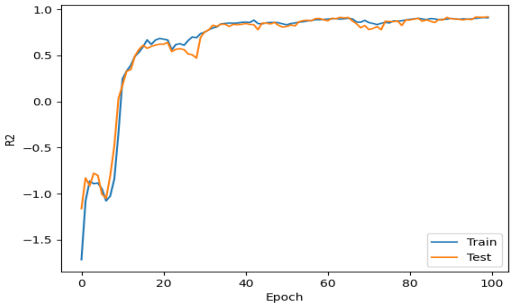
(d) MSE metric data

Figure C.2 Metrics and best epoch data of the quantized LSTM, utilizing deterministic rounding, for *sawtooth*.

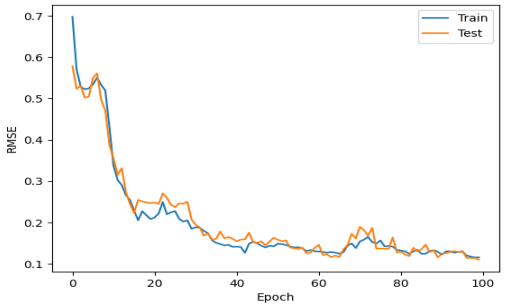


	Training	Validation
MSE (Loss)	0.022737704528935133	0.019592031420500644
RMSE	0.1159713700692972	0.11037570716320673
R2	0.9091666040197237	0.9205938703069095

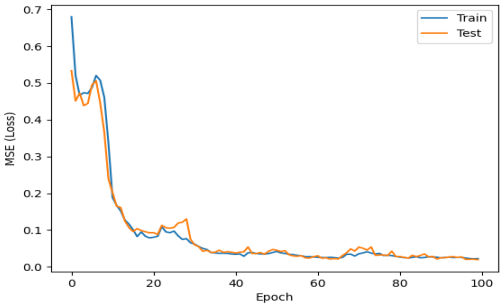
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

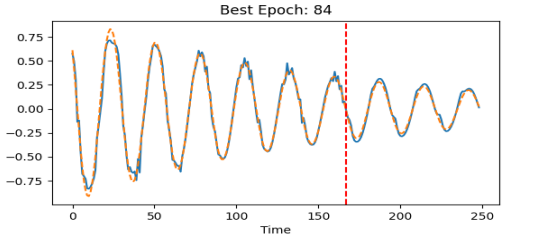


(c) RMSE metric data



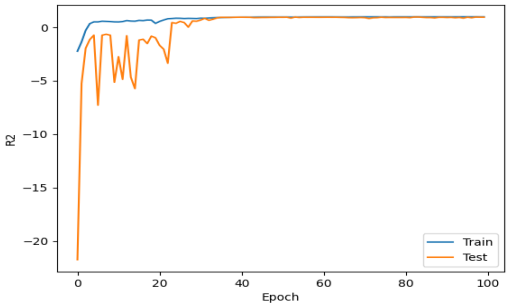
(d) MSE metric data

Figure C.3 Metrics and best epoch data of the quantized LSTM, utilizing deterministic rounding, for *summed cosine waves*.

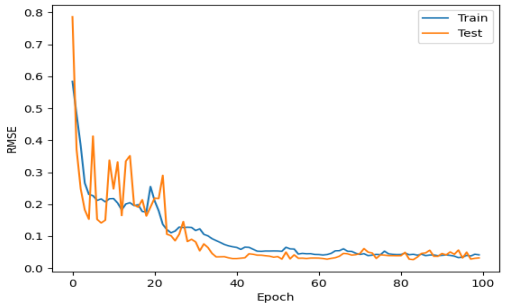


	Training	Validation
MSE (Loss)	0.0032562665516047955	0.0009618690942055161
RMSE	0.04342071052272399	0.02658994870577587
R2	0.9825479890147781	0.9690124886725319

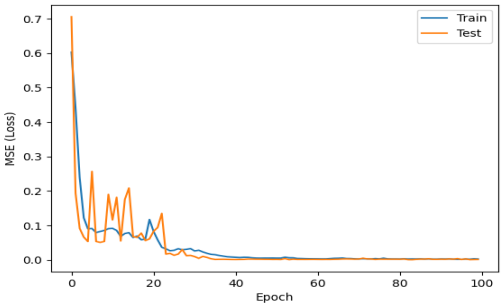
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

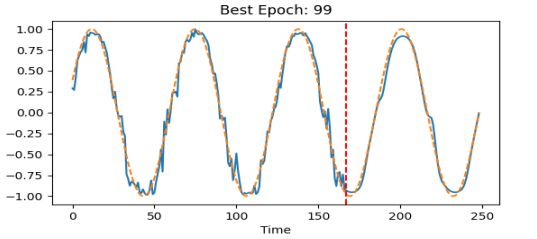


(c) RMSE metric data



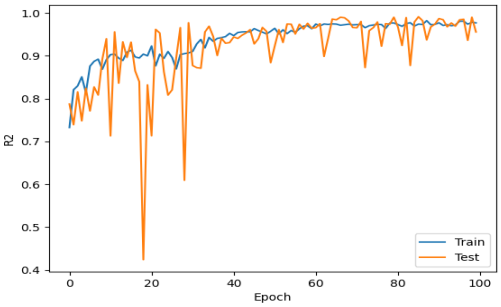
(d) MSE metric data

Figure C.4 Metrics and best epoch data of the quantized LSTM, utilizing deterministic rounding, for *damped harmonic oscillator*.

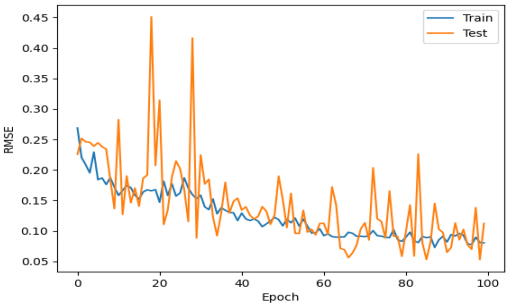


	Training	Validation
MSE (Loss)	0.010789297938617843	0.0048966946940134665
RMSE	0.08107647292140278	0.05289309576990095
R2	0.9781245852941122	0.9900146131375588

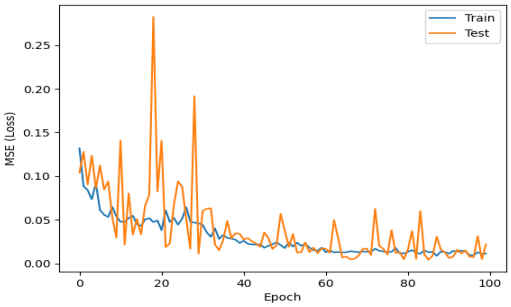
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

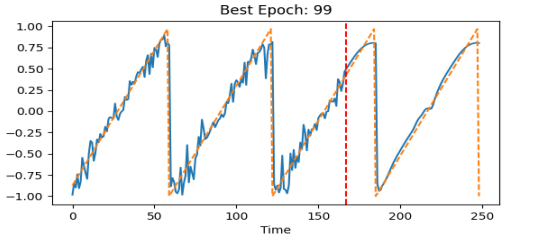


(c) RMSE metric data



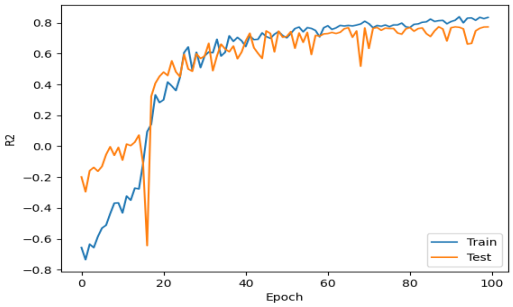
(d) MSE metric data

Figure C.5 Metrics and best epoch data of the quantized LSTM, utilizing stochastic rounding, for *sine*.

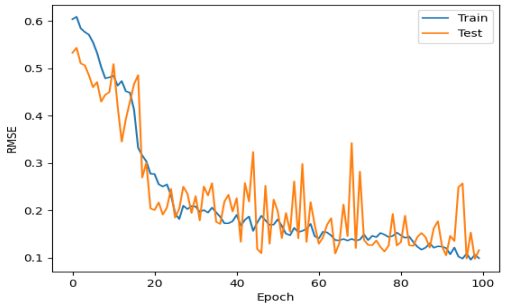


	Training	Validation
MSE (Loss)	0.051186457855968356	0.08232257294435934
RMSE	0.10606305912073101	0.09718257890919076
R2	0.8268832937771072	0.7721417867691672

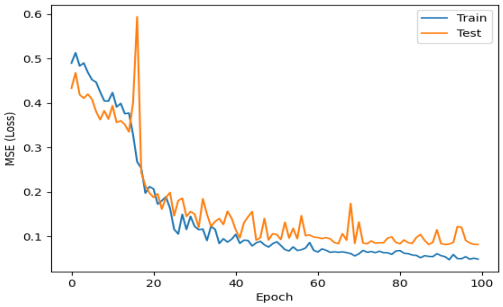
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

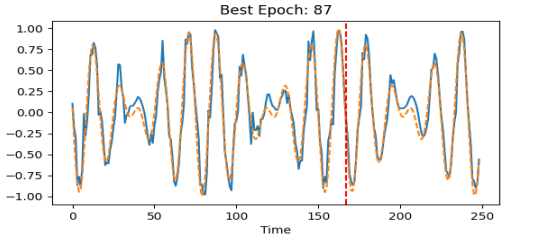


(c) RMSE metric data



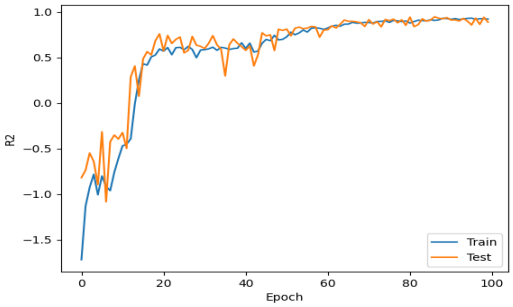
(d) MSE metric data

Figure C.6 Metrics and best epoch data of the quantized LSTM, utilizing stochastic rounding, for *sawtooth*.

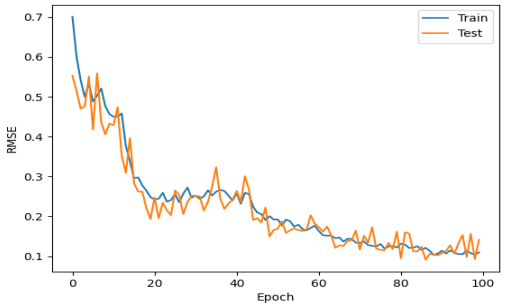


	Training	Validation
MSE (Loss)	0.023235501445838417	0.013021350440060876
RMSE	0.12094321384046863	0.09123117113158266
R2	0.9071779870767382	0.9472247150062894

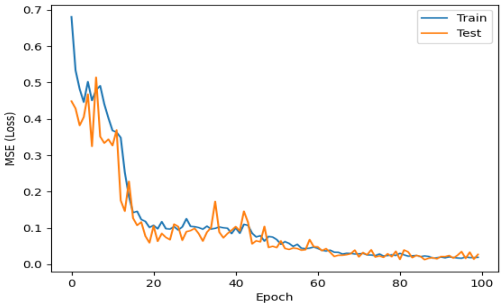
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

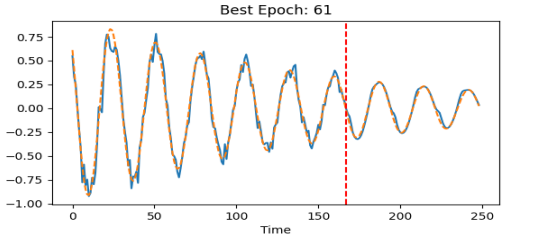


(c) RMSE metric data



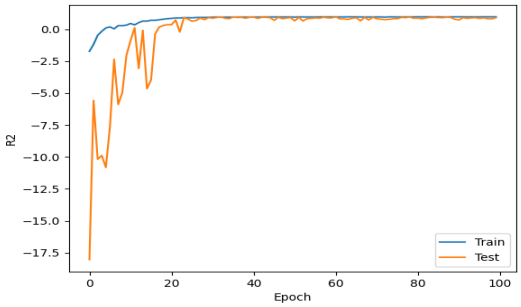
(d) MSE metric data

Figure C.7 Metrics and best epoch data of the quantized LSTM, utilizing stochastic rounding, for *summed cosine waves*.

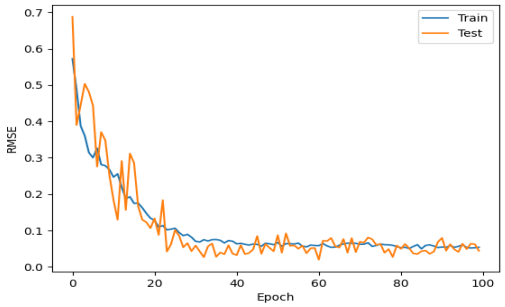


	Training	Validation
MSE (Loss)	0.0059802215012842025	0.0007907884090707862
RMSE	0.05846000407181776	0.01992360424891258
R2	0.9679489102994235	0.9745240127462961

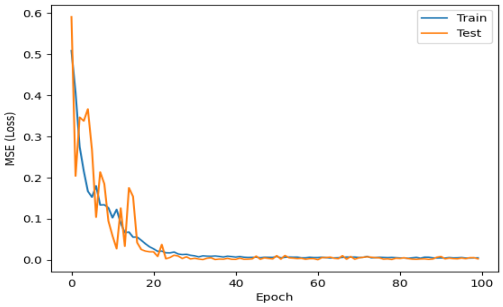
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data



(c) RMSE metric data

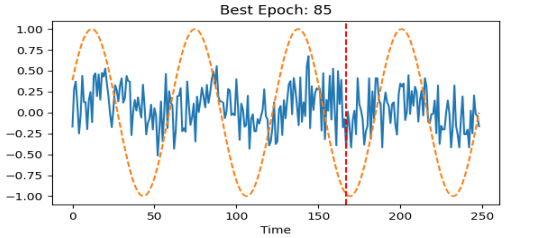


(d) MSE metric data

Figure C.8 Metrics and best epoch data of the quantized LSTM, utilizing stochastic rounding, for *damped harmonic oscillator*.

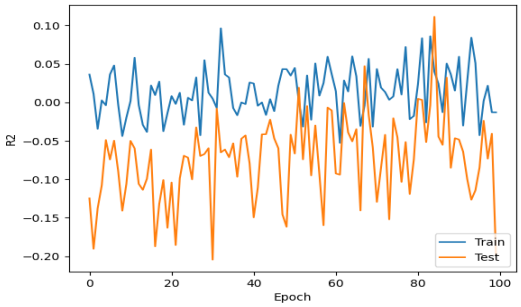
Appendix D

Binary Stochastic Neuron Result Graphics

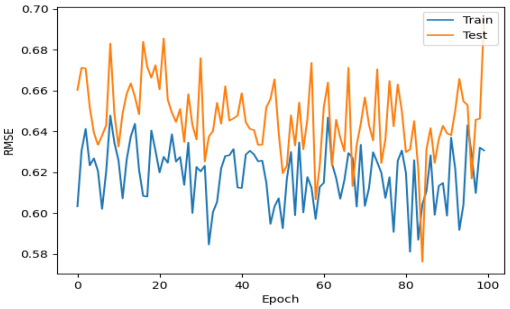


	Training	Validation
MSE (Loss)	0.47350669544992785	0.4357978623441385
RMSE	0.6042908950972752	0.57625434286343
R2	0.039960209838397986	0.11131681240995861

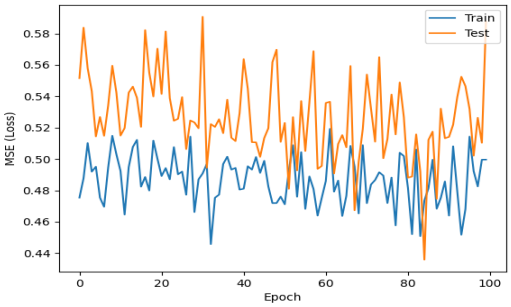
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

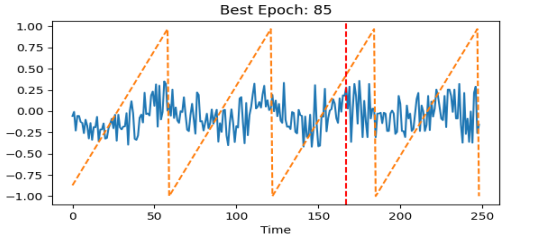


(c) RMSE metric data



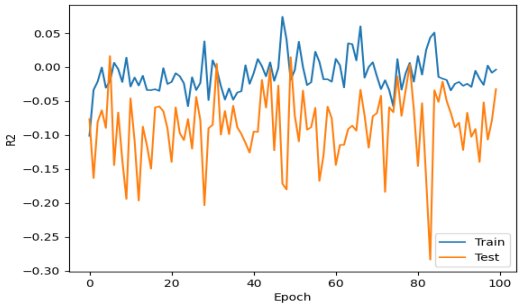
(d) MSE metric data

Figure D.1 Metrics and best epoch data of the stochastic LSTM, utilizing binary stochastic neurons, for *sine*.

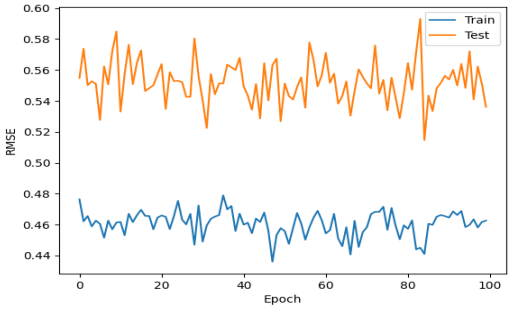


	Training	Validation
MSE (Loss)	0.2806435485084872	0.37364085080422793
RMSE	0.441002394829904	0.5146528724516988
R2	0.050841008041563884	-0.0341894526528288

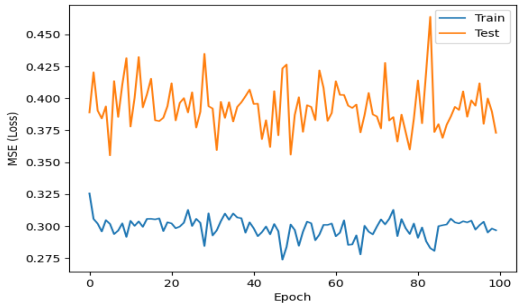
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

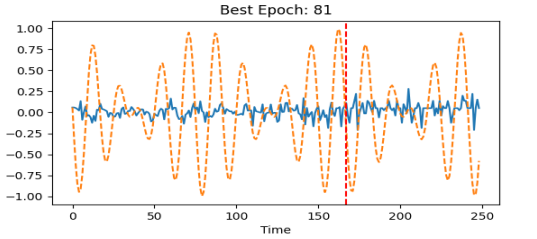


(c) RMSE metric data



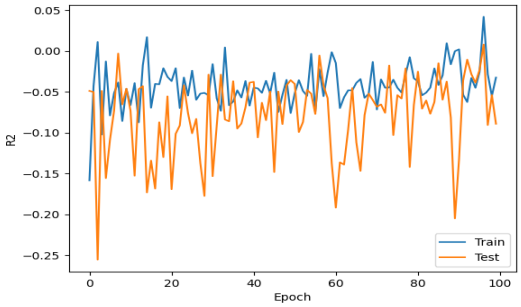
(d) MSE metric data

Figure D.2 Metrics and best epoch data of the stochastic LSTM, utilizing binary stochastic neurons, for *sawtooth*.

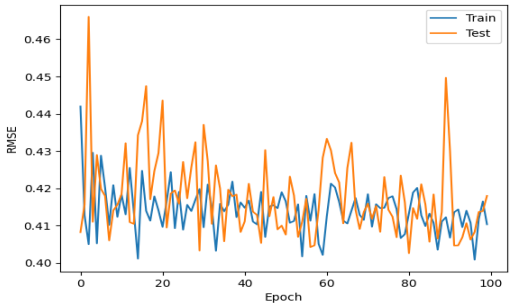


	Training	Validation
MSE (Loss)	0.2596702958409569	0.2529707323408743
RMSE	0.413261300265704	0.40254338739194423
R2	-0.03734019308860059	-0.025285553584633247

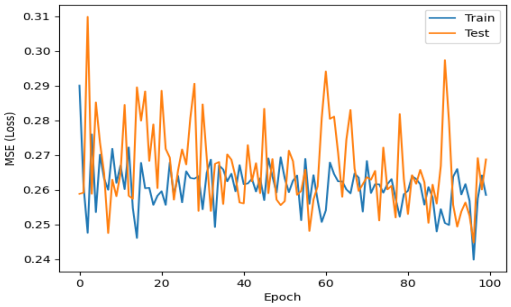
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

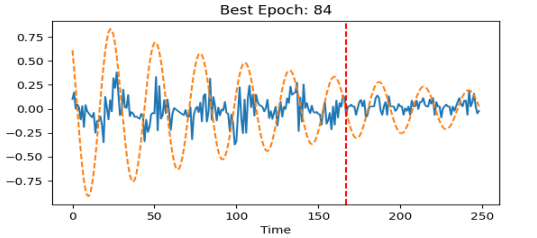


(c) RMSE metric data



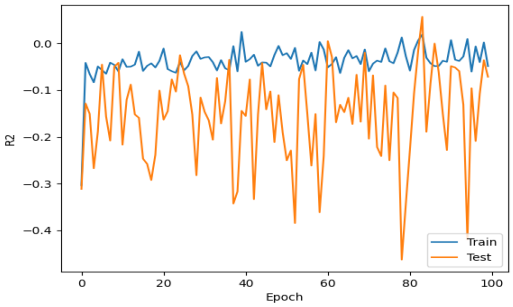
(d) MSE metric data

Figure D.3 Metrics and best epoch data of the stochastic LSTM, utilizing binary stochastic neurons, for *summed cosine waves*.

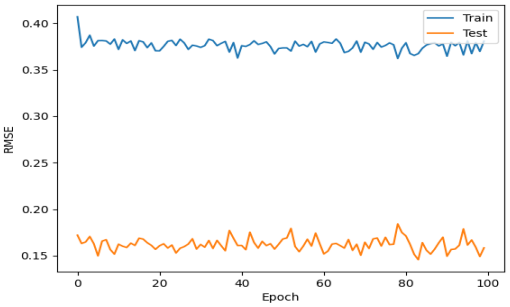


	Training	Validation
MSE (Loss)	0.1829988143614746	0.02926255295479388
RMSE	0.3673481681809096	0.1456540068991715
R2	0.01921502189522073	0.057279523149365685

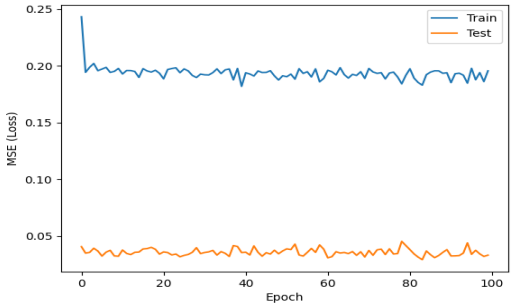
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

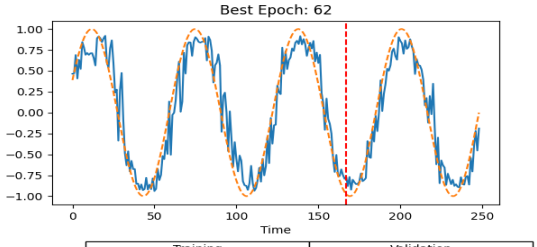


(c) RMSE metric data



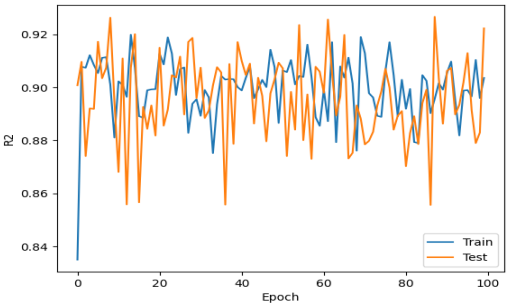
(d) MSE metric data

Figure D.4 Metrics and best epoch data of the stochastic LSTM, utilizing binary stochastic neurons, for *damped harmonic oscillator*.

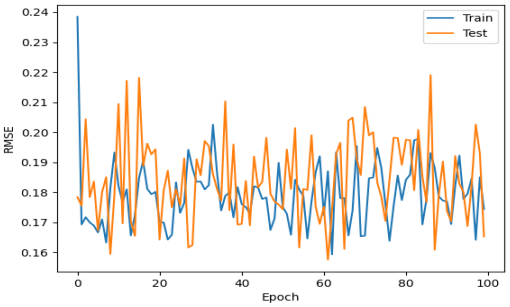


	Training	Validation
MSE (Loss)	0.05561997251650984	0.03650550636477779
RMSE	0.18704865591242945	0.15761371163478036
R2	0.8872299224981269	0.9255576207135665

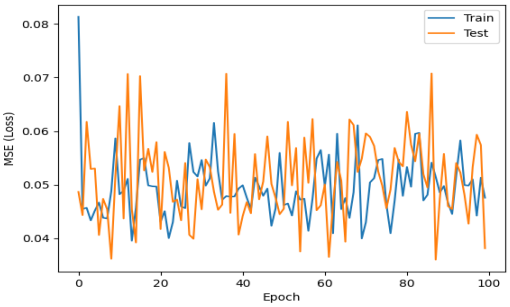
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

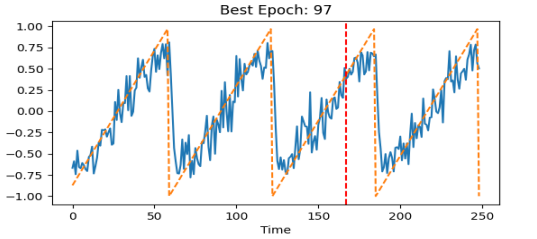


(c) RMSE metric data



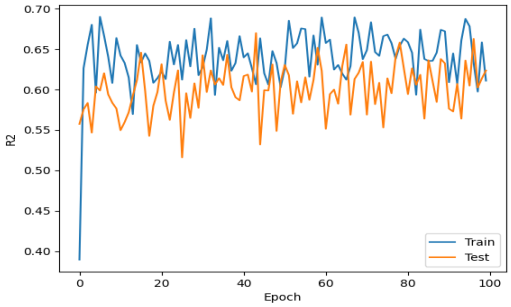
(d) MSE metric data

Figure D.5 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and neuron-based multi-shot inference for *sine*.

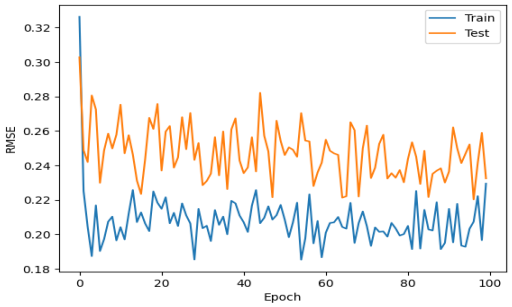


	Training	Validation
MSE (Loss)	0.10862551388847189	0.12188050232651129
RMSE	0.20735550528411154	0.22027747805250408
R2	0.6326197989894959	0.6626505647902239

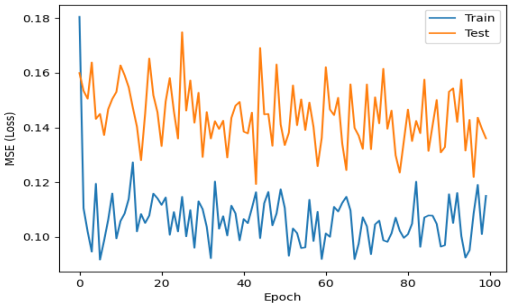
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

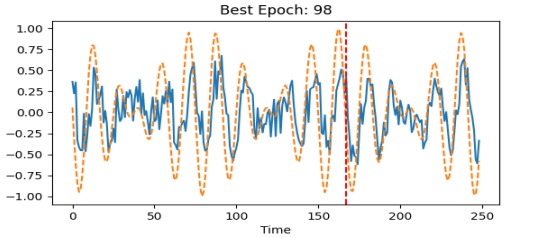


(c) RMSE metric data



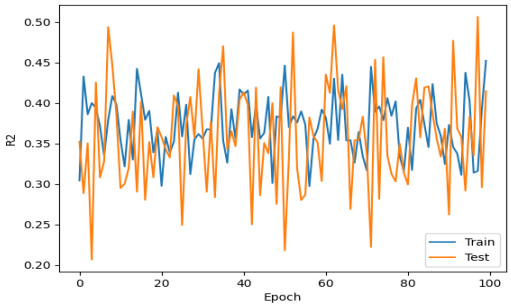
(d) MSE metric data

Figure D.6 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and neuron-based multi-shot inference for *sawtooth*.

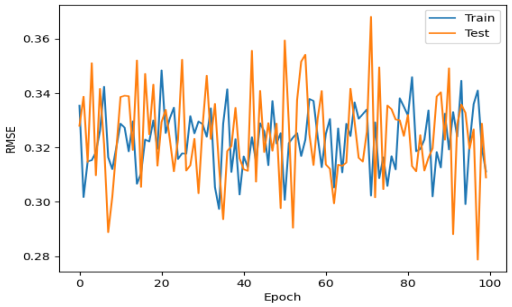


	Training	Validation
MSE (Loss)	0.17122831061470956	0.12173806402506883
RMSE	0.3409879849432036	0.27873137007487747
R2	0.3159710153983577	0.5065979482635321

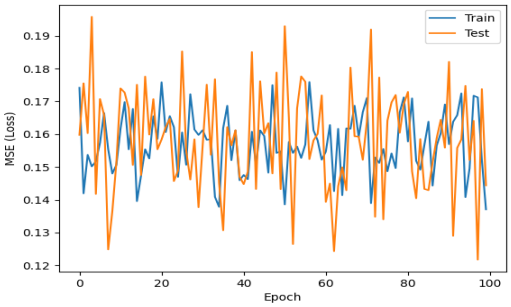
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

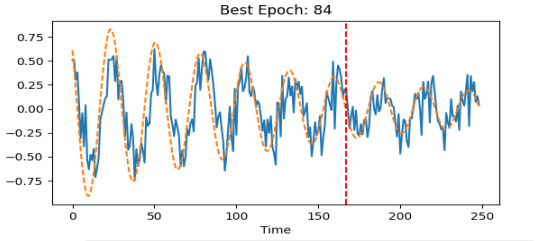


(c) RMSE metric data



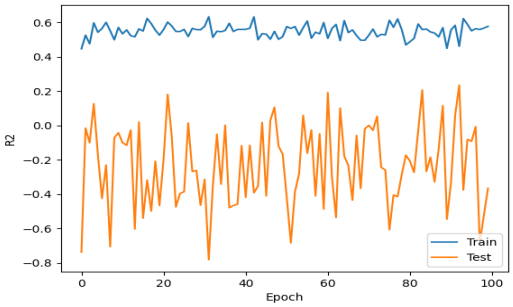
(d) MSE metric data

Figure D.7 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and neuron-based multi-shot inference for *summed cosine waves*.

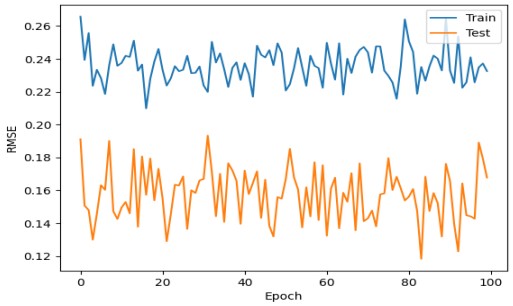


	Training	Validation
MSE (Loss)	0.08230659739902262	0.02463952641061642
RMSE	0.23508563366117563	0.11838302098479513
R2	0.5588765172629786	0.20621464152242353

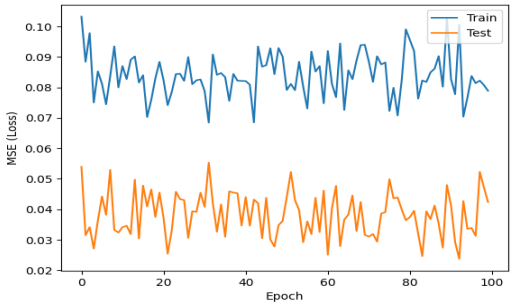
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

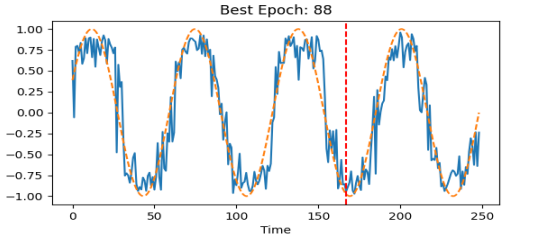


(c) RMSE metric data



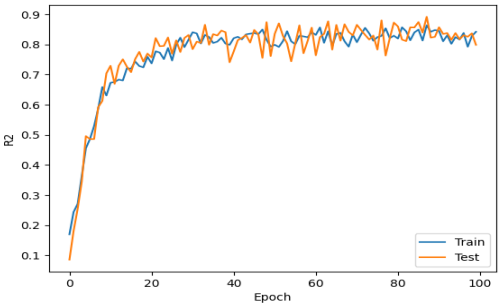
(d) MSE metric data

Figure D.8 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and neuron-based multi-shot inference for *damped harmonic oscillator*.

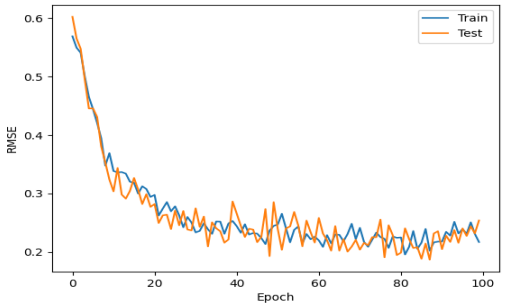


	Training	Validation
MSE (Loss)	0.06707880219588305	0.052854352812603006
RMSE	0.20210174253710902	0.1867818621475095
R2	0.8639970251672245	0.8922188959742509

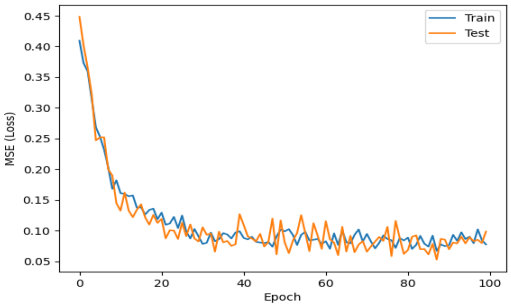
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

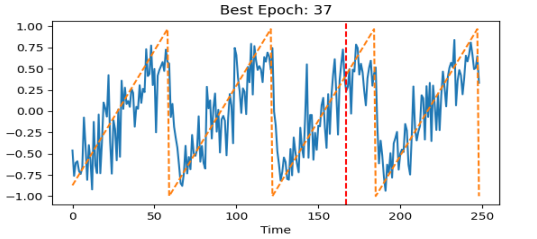


(c) RMSE metric data



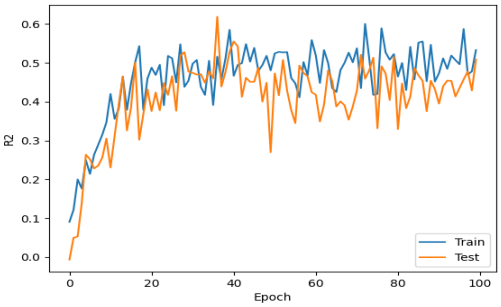
(d) MSE metric data

Figure D.9 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and cell-based multi-shot inference for *sine*.

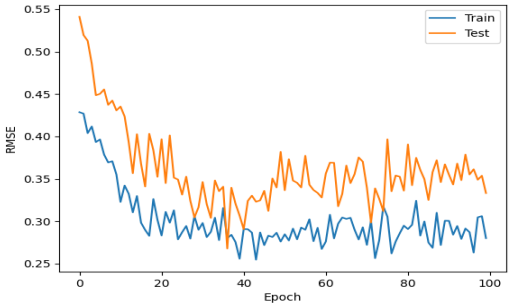


	Training	Validation
MSE (Loss)	0.1430813800242638	0.13779787222493758
RMSE	0.2801249895606458	0.2679097194250478
R2	0.5160872959538383	0.6185933477394273

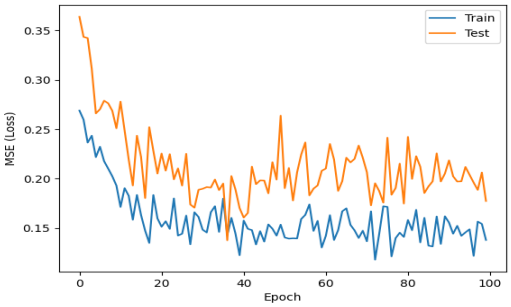
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

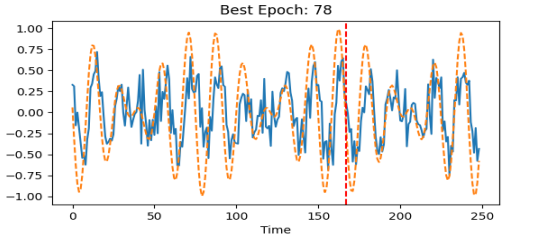


(c) RMSE metric data



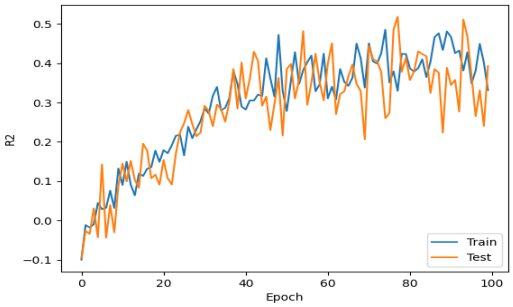
(d) MSE metric data

Figure D.10 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and cell-based multi-shot inference for *sawtooth*.

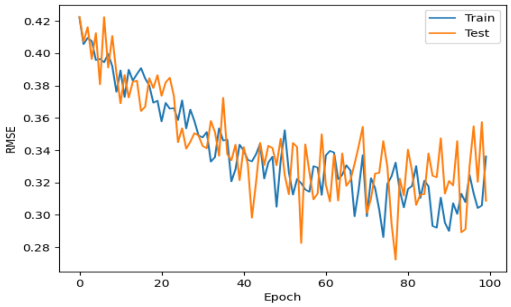


	Training	Validation
MSE (Loss)	0.16782374038310993	0.11894006001515559
RMSE	0.33238933023654416	0.27225579771191627
R2	0.3295717144309277	0.5179382051528956

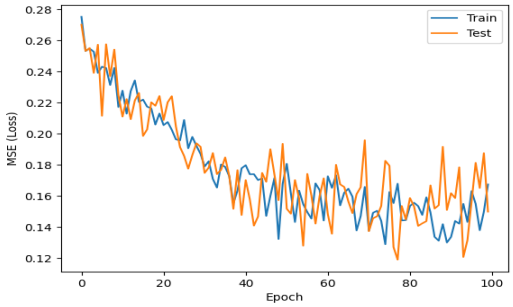
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

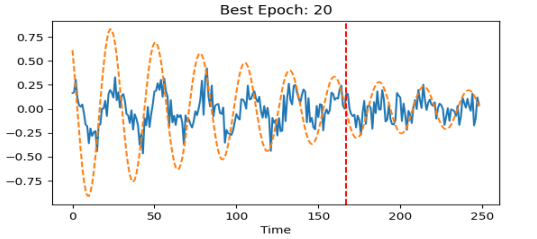


(c) RMSE metric data



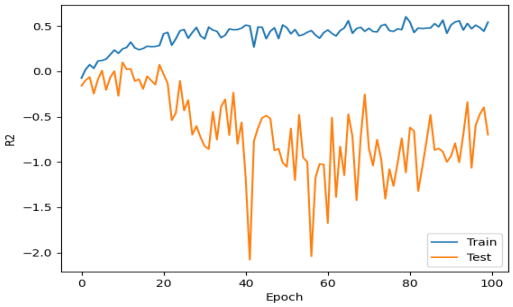
(d) MSE metric data

Figure D.11 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and cell-based multi-shot inference for *summed cosine waves*.

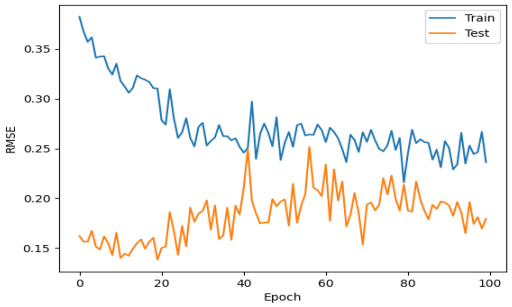


	Training	Validation
MSE (Loss)	0.13341432658562855	0.02876114597964798
RMSE	0.31031137584733715	0.13820005732721857
R2	0.2849638516199219	0.07343281720528727

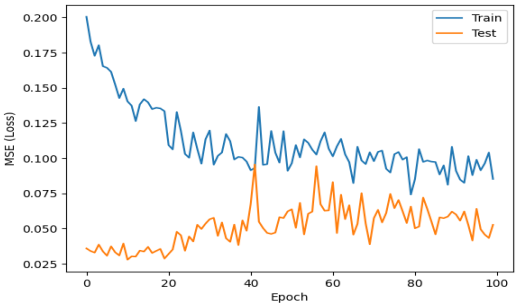
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data



(c) RMSE metric data

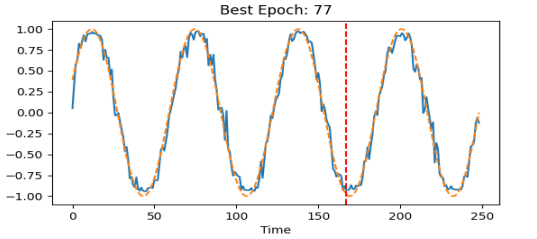


(d) MSE metric data

Figure D.12 Metrics and best epoch data of the stochastic LSTM utilizing binary stochastic neurons and cell-based multi-shot inference for *damped harmonic oscillator*.

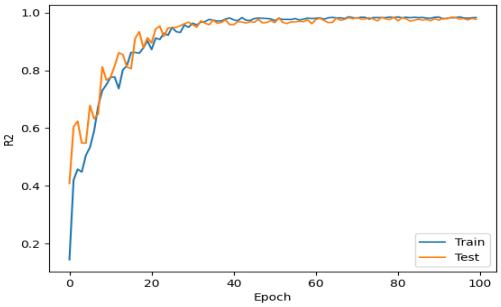
Appendix E

Quantized MAC Result Graphics

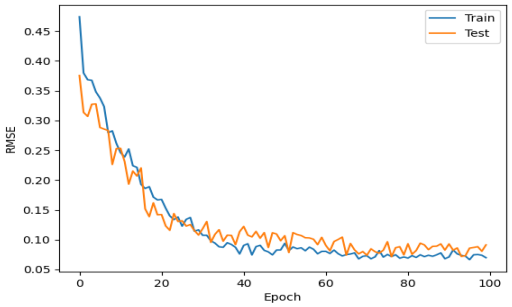


	Training	Validation
MSE (Loss)	0.008364373316062794	0.007880513801444469
RMSE	0.07169273075063166	0.07150882560327494
R2	0.9830411453938239	0.9839299805482188

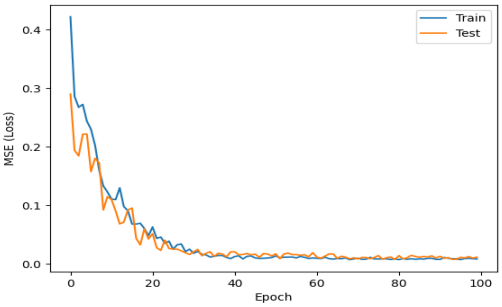
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

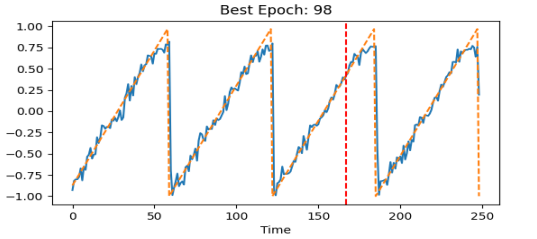


(c) RMSE metric data



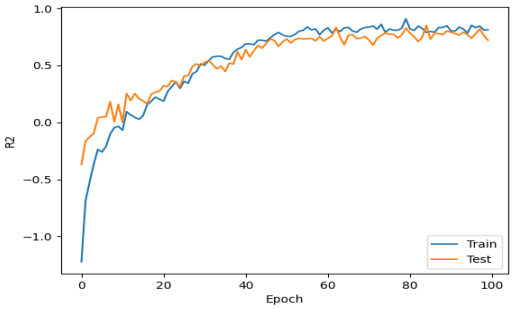
(d) MSE metric data

Figure E.1 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs for *sine*.

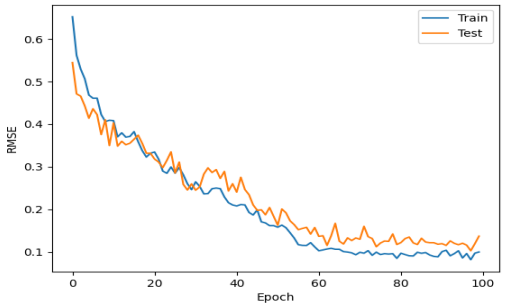


	Training	Validation
MSE (Loss)	0.045826362665860036	0.06502333703854256
RMSE	0.08187285622644554	0.10289483853239653
R2	0.8450115656525272	0.8200238298440631

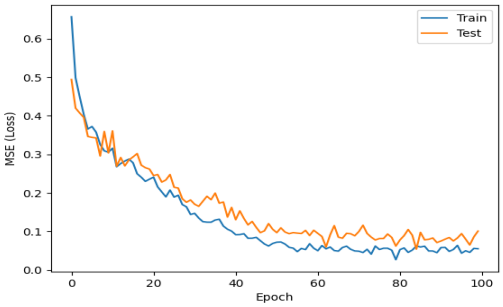
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

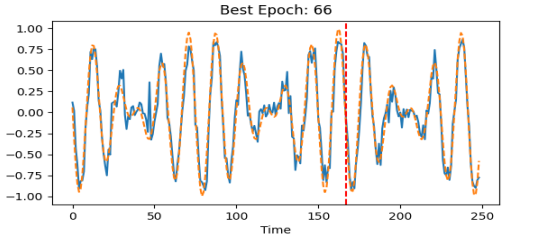


(c) RMSE metric data



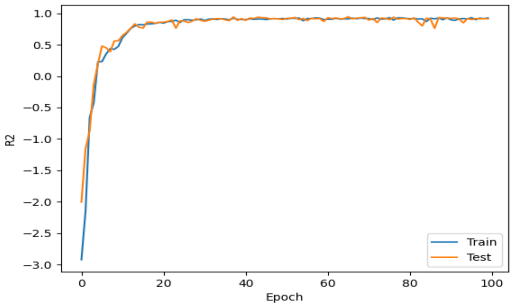
(d) MSE metric data

Figure E.2 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs for *sawtooth*.

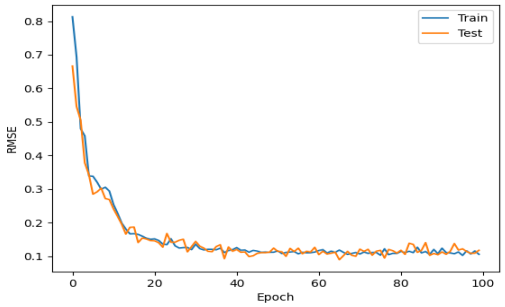


	Training	Validation
MSE (Loss)	0.022051227250079845	0.013445183619046903
RMSE	0.11836493328552149	0.08966626383531046
R2	0.9119089680267136	0.9455069272150967

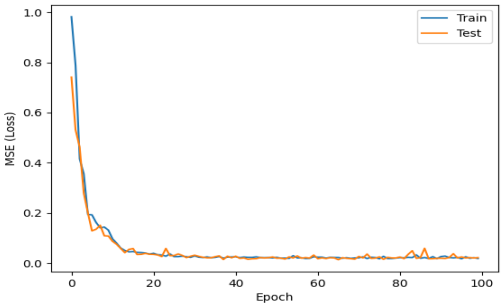
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

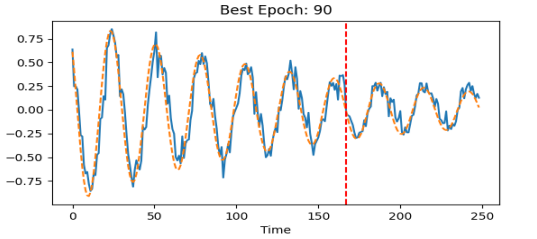


(c) RMSE metric data



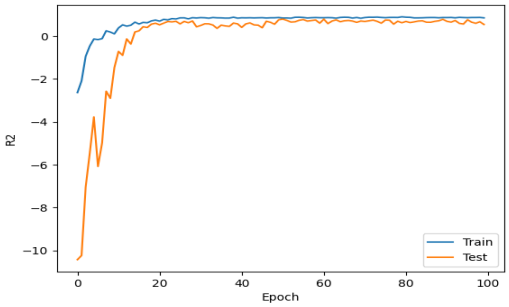
(d) MSE metric data

Figure E.3 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs for *summed cosine waves*.

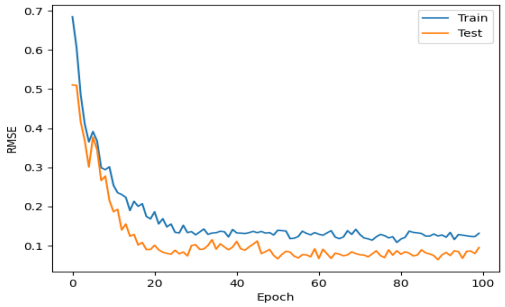


	Training	Validation
MSE (Loss)	0.024917900213675674	0.006796939637015473
RMSE	0.12476409033714109	0.06423324313801841
R2	0.8664521281148148	0.7810302407438279

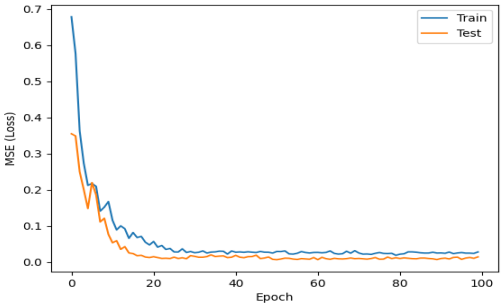
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

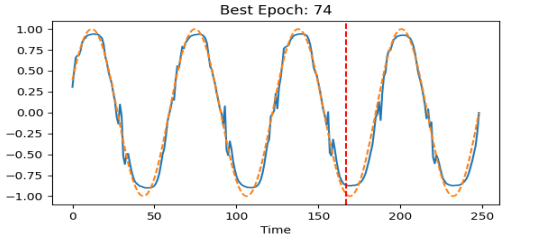


(c) RMSE metric data



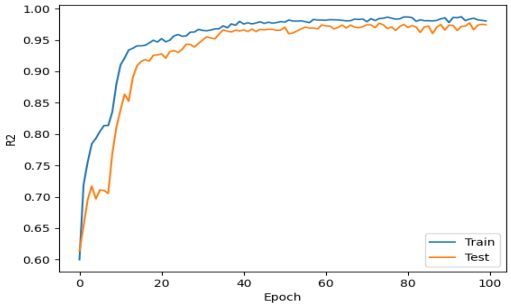
(d) MSE metric data

Figure E.4 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs for *damped harmonic oscillator*.

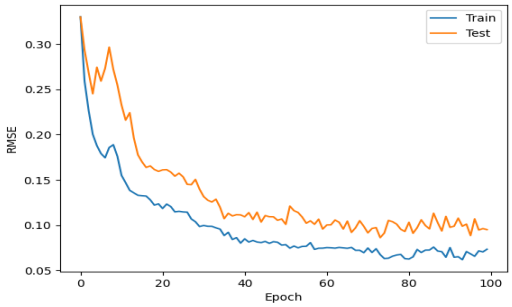


	Training	Validation
MSE (Loss)	0.007723815114678416	0.011351173364064418
RMSE	0.06740335510274173	0.08618993717283387
R2	0.9843398838639505	0.9768525782255948

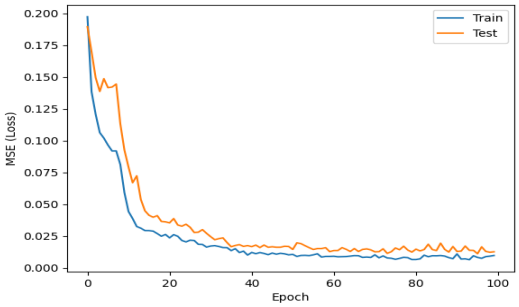
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

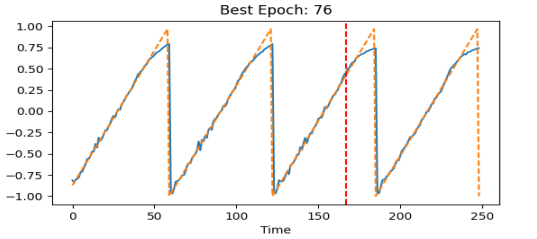


(c) RMSE metric data



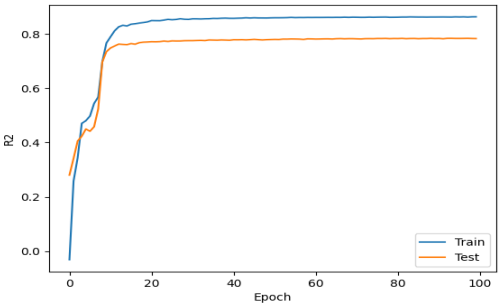
(d) MSE metric data

Figure E.5 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and MAC-based multi-shot inference for *sine*.

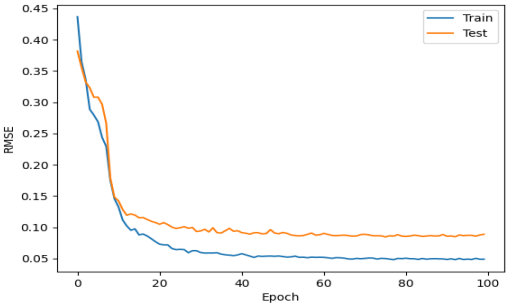


	Training	Validation
MSE (Loss)	0.04086787075513999	0.07829619116578916
RMSE	0.05003493641883414	0.08461518977413757
R2	0.8617815830237648	0.7832862897291302

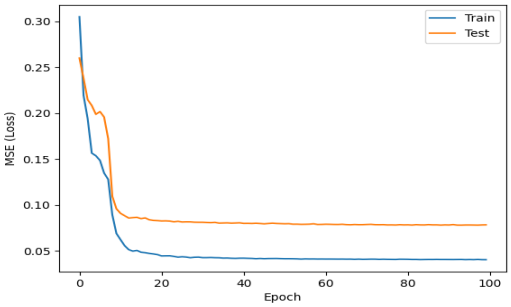
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

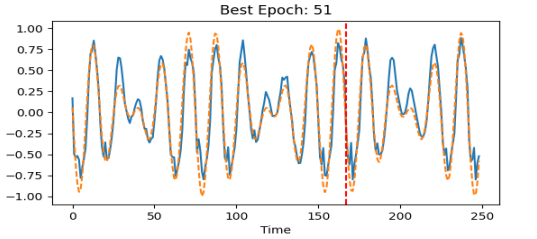


(c) RMSE metric data



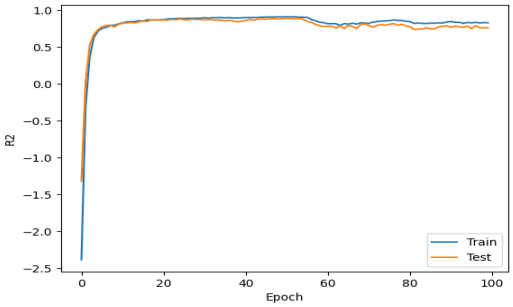
(d) MSE metric data

Figure E.6 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and MAC-based multi-shot inference for *sawtooth*.

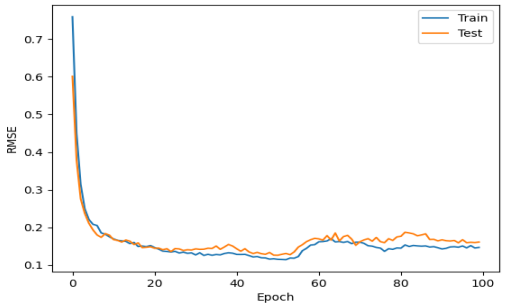


	Training	Validation
MSE (Loss)	0.02307470406342252	0.027852315171601406
RMSE	0.11548623594684931	0.12626020658690218
R2	0.9078203462159815	0.8871150978017115

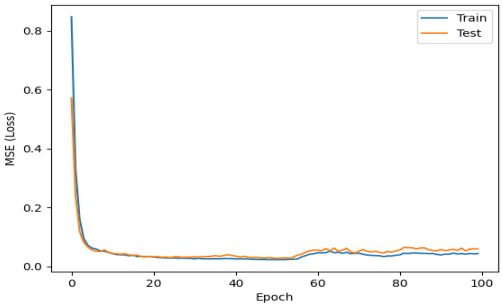
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

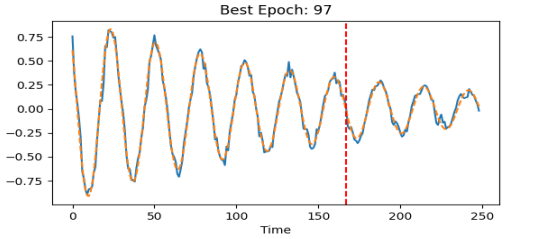


(c) RMSE metric data



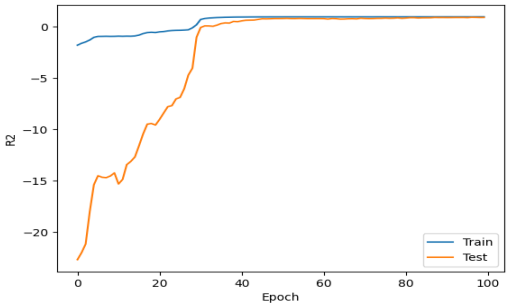
(d) MSE metric data

Figure E.7 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and MAC-based multi-shot inference for *summed cosine waves*.

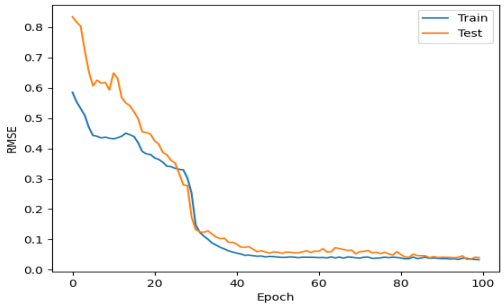


	Training	Validation
MSE (Loss)	0.0025987168771086236	0.001820312608929567
RMSE	0.038172231087791976	0.03422204993121797
R2	0.9860721366730774	0.9413569289952818

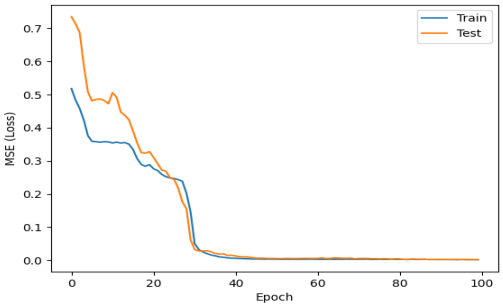
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

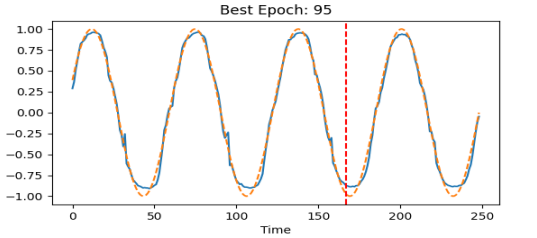


(c) RMSE metric data



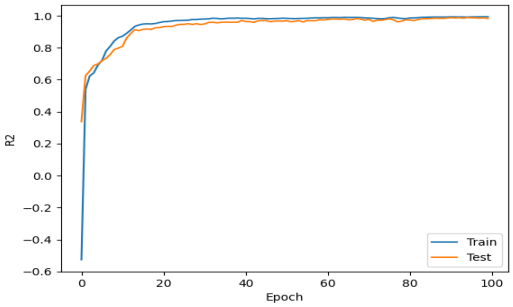
(d) MSE metric data

Figure E.8 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and MAC-based multi-shot inference for *damped harmonic oscillator*.

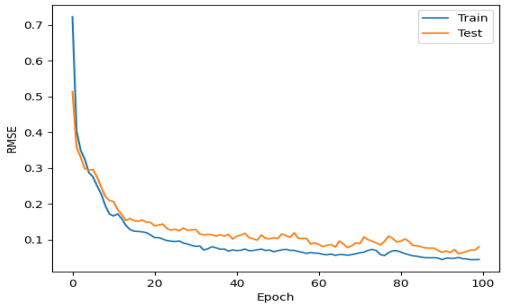


	Training	Validation
MSE (Loss)	0.004040671842856491	0.005301421972989382
RMSE	0.05086039158369966	0.0612387019905909
R2	0.9918074954685872	0.9891892893820674

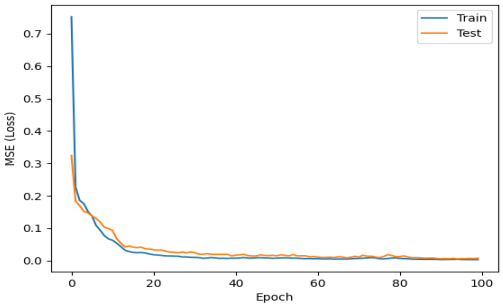
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

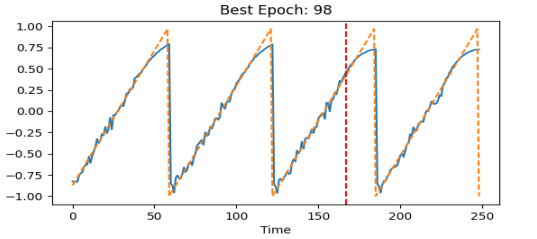


(c) RMSE metric data



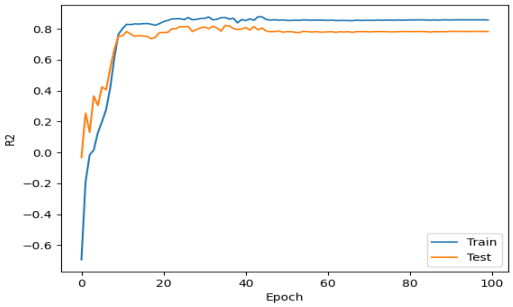
(d) MSE metric data

Figure E.9 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and cell-based multi-shot inference for *sine*.

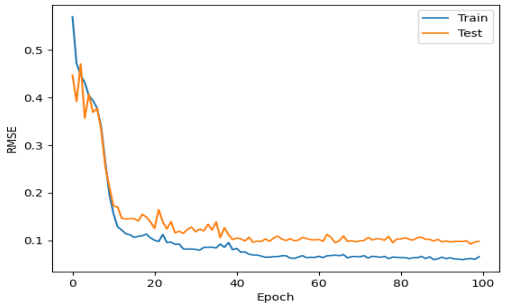


	Training	Validation
MSE (Loss)	0.04178635148369999	0.07815663858895186
RMSE	0.06233992578116566	0.09238498751818536
R2	0.8586752075268527	0.783672553176355

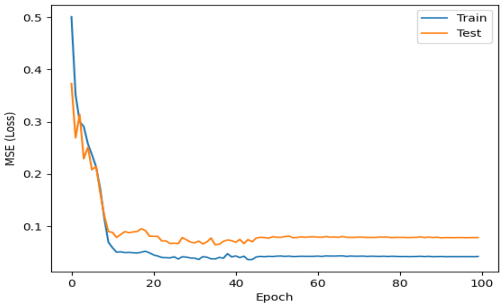
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

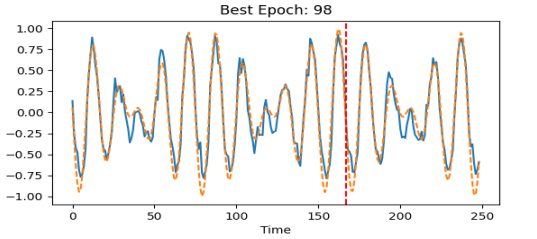


(c) RMSE metric data



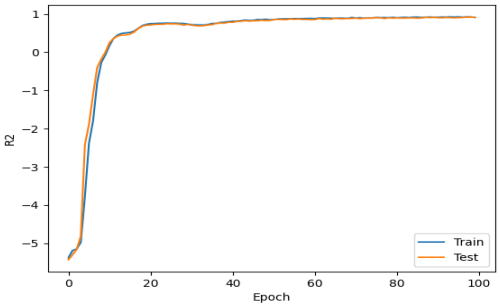
(d) MSE metric data

Figure E.10 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and cell-based multi-shot inference for *sawtooth*.

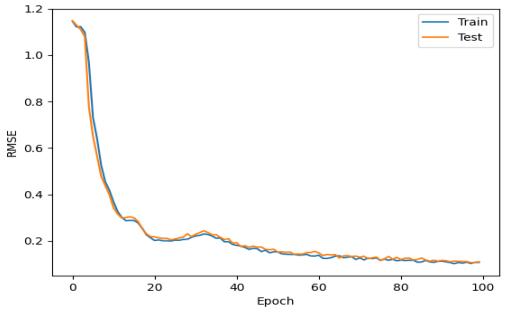


	Training	Validation
MSE (Loss)	0.018137275246001994	0.020355832161438166
RMSE	0.10276846486167943	0.10658694088040732
R2	0.927544563598016	0.9174982004709017

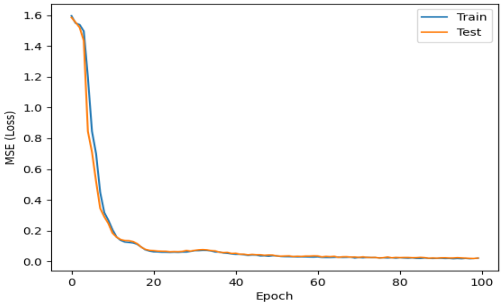
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data

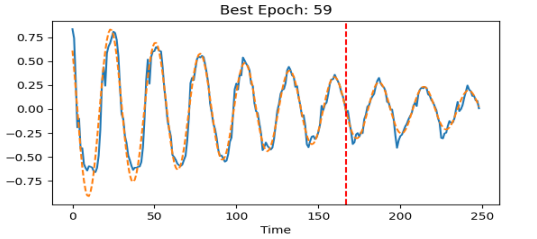


(c) RMSE metric data



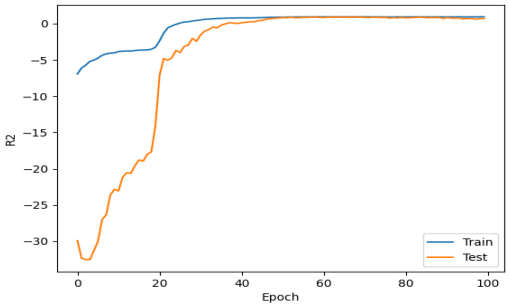
(d) MSE metric data

Figure E.11 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and cell-based multi-shot inference for *summed cosine waves*.

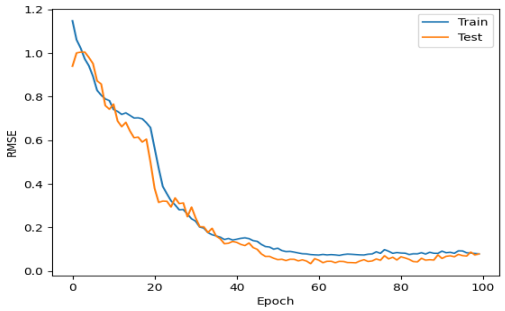


	Training	Validation
MSE (Loss)	0.011891682129003427	0.0024092941040527644
RMSE	0.07586219930743565	0.03400556623447741
R2	0.936266345565029	0.9223823399771429

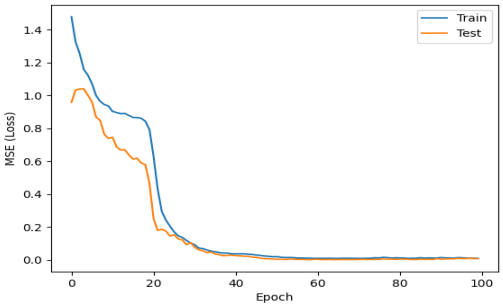
(a) data and output from best epoch recorded during training phase



(b) R^2 metric data



(c) RMSE metric data



(d) MSE metric data

Figure E.12 Metrics and best epoch data of the stochastic LSTM utilizing quantized MAC outputs and cell-based multi-shot inference for *damped harmonic oscillator*.