UNIVERSITY OF CALIFORNIA

Los Angeles

Unitary Neural Networks

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Electrical and Computer Engineering

by

Hao-Yuan Chang

2022

ABSTRACT OF THE DISSERTATION

Unitary Neural Networks

by

Hao-Yuan Chang

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Los Angeles, 2022

Professor Kang L. Wang, Chair

This doctoral dissertation is a comprehensive study on a novel method based on unitary synaptic weights to construct intrinsically stable neural systems. By eliminating the need to normalize neural activations, unitary neural networks deliver faster inference speeds and smaller model sizes while maintaining competitive accuracies for image recognition. In addition, unitary networks are drastically more robust against adversarial attacks in natural language processing systems because unitary weights are resilient to small input perturbations. The last portion focuses on a small demo that implements unitary neural nets in quantum computing. With the comprehensive performance evaluation in classical machine learning, the rigorous framework in mathematics, and the exploration of quantum computing, this dissertation establishes a solid foundation for unitary neural networks in the future of deep learning.

The dissertation of Hao-Yuan Chang is approved.

Lieven Vandenberghe

Jonathan Chau-Yan Kao

Guido F. Montufar Cuartas

Kang L. Wang, Committee Chair

University of California, Los Angeles

2022

Table of Contents

Table of Figures

List of Tables

Acknowledgments

The author would like to thank Professor Robert N. Schwartz for his valuable discussions and the National Defense Science and Engineering Graduate Fellowship for their support. Additionally, Chapter 2 is a version of our published work [1], in which Kang L. Wang is the principal investigator (PI). The final authenticated publication is available online at https://doi.org/10.1007/978-3-030-86340-1_14.

Vita

Hao-Yuan Chang obtained his Bachelor of Science with summa cum laude and his Master of Science both in Electrical Engineering from the University of California, Los Angeles (UCLA). His research focuses on deep learning for natural language processing and computer vision. He is the recipient of the National Defense Science and Engineering Graduate Fellowship and the UCLA Dissertation Year Fellowship. Selected publications and patents include:

Hao-Yuan Chang and Kang L. Wang, “Deep Unitary Convolutional Neural Networks,” in *Artificial Neural Networks and Machine Learning – ICANN 2021*, Cham, 2021, pp. 170–181. doi: 10.1007/978-3-030-86340-1_14.

Kang L. Wang and Hao-Yuan Chang, “Self-organized critical CMOS circuits and methods for computation and information processing,” United States Patent US10147045B2, Dec. 04, 2018. [Online]. Available: https://patents.google.com/patent/US10147045

Chapter 1 — Background

The transition to quantum computing.

1.1. Motivation

The twentieth century was full of hardware innovations: Vacuum tubes and bipolar-junction transistors enabled digital computing. Metal-oxide-semiconductors field-effect transistors (MOSFET) paved the way for the very-large-scale integration (VLSI) of logic devices and led to the invention of microprocessors in 1971. Since then, computational speed and capacity have grown exponentially, adhering to Dennard's law for feature size scaling [2], [3] and Moore's law for economic scaling [4]. This massive computing power allowed computer scientists to build increasingly accurate and large machine learning models. The most remarkable example is the renaissance of neural networks a decade ago, driven by the adoption of graphics computing units (GPU) in parallelizing deep neural network computations, leading to the revival of artificial intelligence (AI) and machine learning (ML).

Today, we stand at a pivotal point of the technology curve for semiconductors: As transistor scaling approaches its physical limit, improvements slow down. For any technology, there is a point of saturation, where all the possible improvements to the given technology have been made. It's a place of incredible achievements because, at this point, we have discovered the most optimal way to engineer transistors. But this is also the point of stagnation, which can only be surpassed by an orthogonal technology that disrupts the field. The rise and fall of technology curves are quotidian in the computer industry [5]. We believe that transistors have reached the confines of physics, and this disruptive technology will be quantum.

A quantum system's computing power scales exponentially with the number of bits, unlike its classical counterparts that scale linearly. This property makes quantum computing an attractive alternative to transistor scaling, enabling the next-generation deep learning architectures for big data. Quantum gates are unitary transforms because they conserve energy, but most neural networks today are non-unitary. To address this research gap, we thoroughly investigate the possibility of imposing unitarity on parts of the neural net and study the benefits of unitary weights in deep learning. As you will see in the following chapters, unitary neural nets not only improve the speed and robustness of modern machine learning systems but also establish a steppingstone for the quantum transition.

1.2. Mathematics of Unitarity

In mathematics, a unitary matrix (**U**) is an n-by-n square matrix that satisfies this special relationship:

$$\boldsymbol{U}^{\dagger}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^{\dagger} = \boldsymbol{I}, \tag{1}$$

where $\boldsymbol{U}$ can be real or complex, $\boldsymbol{I}$ is the identity matrix, and † represents the Hermitian adjoint. When $\boldsymbol{U}$ is real, it is called an orthogonal matrix, a subclass of the unitary matrix, and $\boldsymbol{U}^{\dagger}$ changes to $\boldsymbol{U}^{T}$, the transposition of $\boldsymbol{U}$.

The unitary matrix has a special property that conserves the Euclidean norm of the input vector. To prove this conservation law, we take the Euclidean norm of the output and compare it to the norm of the input:

$$\|\boldsymbol{U}\bar{x}\|_2 = (\boldsymbol{U}\bar{x})^{\dagger}(\boldsymbol{U}\bar{x}) = \bar{x}^{\dagger}\boldsymbol{U}^{\dagger}\boldsymbol{U}\bar{x} = \bar{x}^{\dagger}\bar{x} = \|\bar{x}\|_2, \tag{2}$$

where $\|\cdot\|_2$ denotes the Euclidean norm. This conservation law is the main theme of this dissertation: In classical systems, it stabilizes the neural signals, making the network faster and more robust. In quantum systems, it conserves the energy of a quantum system [6], [7] and makes the network ready to be implemented on dissipation-less quantum computers. We will rigorously quantify the performance improvements in the following three chapters.

For the rest of this dissertation, we assume $\boldsymbol{U}$ is real. This simplification is possible because we represent the weights and the activation values of neural networks using real numbers. In our case, the Hermitian adjoint in the previous two equations reduces to the matrix transpose hereafter.

1.3. Dissertation Outline

Chapter 1 establishes the premise and motivations of our research. To provide a comprehensive perspective on unitary neural networks, we first identify their benefits and drawbacks in classical computing when parts of the neural weights are constrained to be unitary for various network architectures and applications. **Chapter 2** shows that unitary neural nets are stable by design and thus do not require explicit normalization after each layer. The elimination of normalization speeds up deep neural nets significantly. The experiments are conducted with a state-of-the-art convolutional neural network (CNN) on image recognition datasets. Furthermore, **Chapter 3** demonstrates that unitary neural nets are drastically more robust against adversarial attacks because unitary weights prevent small perturbations from growing larger as signals propagate across the layers in a deep neural net. We demonstrate this unique advantage by performing natural language processing using transformer-based neural architectures. **Chapter 4** studies a small demo using a hybrid quantum neural net (QNN). We observe a direct correlation between the intrinsic dimensionality of the input signals and the required number of quantum layers in a QNN. Lastly, **Chapter 5** summarizes our findings and suggests a list of future work in this research direction.

Chapter 2 — Unitary Convolutional Neural Net for Faster Computer Vision

Unitarity eliminates the need to normalize.

2.1. Introduction

A. The Problem of Instability in Deep Neural Nets

Recent advancements in semiconductor technology [8] have enabled neural networks to grow significantly deeper. This abundant computing power enabled computer scientists to drastically increase the depths of neural networks from the 7-layer LeNet network [9] to the 152-layer contest-wining ResNet architecture [10]. More layers usually lead to higher recognition accuracy because neural networks make decisions by drawing decision boundaries in the high-dimensional space [11]. A decision boundary is a demarcation in the feature space that separates the different output classes. The more layers the network has, the more precise these boundaries can be in the high-dimensional feature space; thus, they can achieve higher recognition rates [12]. However, deep networks often fail to train properly due to poor convergence as explained below.

There are many reasons why a deep network fails to train [13], and the problem that unitary neural nets address is the instability of the forward pass, in which neural activations either saturate to infinity or diminish to zero and thus prevent the training algorithm from converging. To train a neural net, the error observed at the output is backpropagated to update each synaptic weight in the previous layers. In other words, the neural net is a composite function (f) with the weights as internal parameters, mapping an input to an output. The goal of training is to find the best weights for producing the desired output, and we do so by taking the derivative of f against the weight variables using the chain rule. During this process, the error signals travel backward through the network, multiplied by the activation at each neuron; hence, this training procedure is called backpropagation [12]. Depending on the eigenvalues of the synaptic weight matrices [14], neural signals may grow or attenuate as they travel across neural layers in the forward pass when unbounded activation functions[1] are used. If the activation is either extremely large or small; in this case, the weight update will scale proportionally during the backward pass, resulting in either a massive or a tiny step. Consequently, convergence is poor during training.

In short, vanishing and exploding activations occur when the neural signals are not normalized, and the backpropagated gradients either saturate or die out during network training [16]. Although other schemes such as batch normalization [17], learning rate tuning [18], and gradient highways [10] can mitigate the issue, none of these methods eliminate the core problem—the weight matrices have eigenvalues that are larger or smaller than one. Furthermore, most normalization methods have inference time penalties because of the additional steps to compute and rescale the neural signals to the norm. In this section, we aim to devise a way to fundamentally fix the exploding and vanishing activation problem without slowing down the inference speed.

B. __Our Proposed Solution__

Our proposed solution (Fig. 1) is to eliminate the need to normalize the neural signals after each layer by constraining the weight matrices, $\boldsymbol{W}$, to be unitary. Unitary matrices[2] represent rotations in the n-dimensional space; hence, they preserve the norm (i.e., the amplitude)

[1] Unbounded activation function is common in deep learning. For example, the rectified linear unit (ReLU) is a popular nonlinearity in many neural architectures due to its computational simplicity [15].

[2] Unitary matrices can have complex values. When the matrices only contain real components, they are called orthogonal matrices, which is a subset of unitary matrices, and our proposal works in both cases. The eigenvalues of a unitary matrix have modulus 1.

of the input vector. With this unique property, unitary networks can maintain the neural signal strengths without explicit normalization. This technique allows the designers to eliminate the networks' normalization blocks and make inference faster.

We aim to engineer a way to constrain the weights to be unitary. To achieve this, we leverage the previously reported framework for constructing orthogonal matrices in recurrent neural networks using Lie algebra [19], which we will explain briefly in Sect. 2.2A below. Unlike other approximation methods, this framework guarantees strictly unitary matrices; however, it is currently limited to square matrices. Our main contribution is that we found a way (Sect. 2.2B) to extend the unitary framework based on Lie algebra to weight matrices of any shape. By doing so, we expand the applicability of this framework from recurrent neural networks with square weight matrices to any neural network structures, drastically increasing its usefulness in state-of-the-art network architectures.

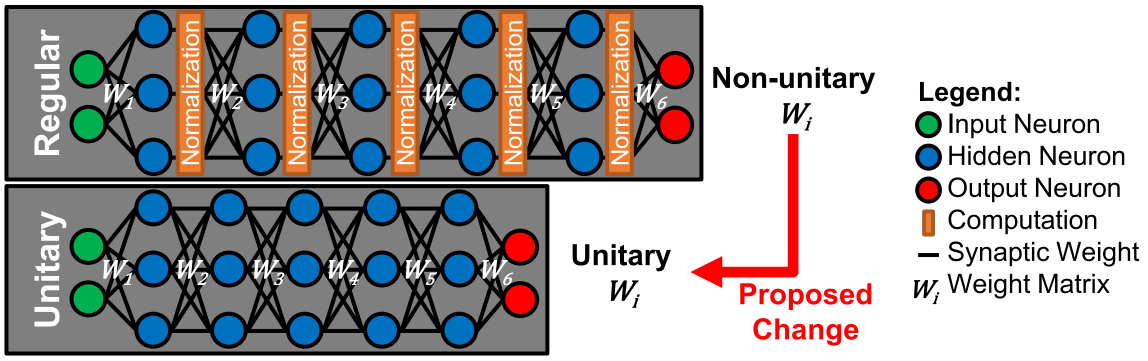

Fig. 1. A unitary neural network for mitigating exploding and vanishing activations.[3] Deep neural nets are unstable systems by default because non-unitary synaptic weights (non-unitary $\boldsymbol{W_i}$) can arbitrarily scale the activations (i.e., neurons'

[3] Figure from our published work [1].

output signals); thus, normalization is required to maintain signal stability in a regular neural net (top). We propose to use unitary constraints on the synaptic weights (unitary $\boldsymbol{W_i}$) to regulate the activations with the norm preserving property of unitary matrices (bottom). By doing so, we can eliminate the computationally expensive normalization steps and improve the network's inference speed.

C. <u>Literature Review</u>

Lie algebra is not the only way to construct unitary matrices. Researchers have explored many options to construct unitary weights for RNNs, including eigendecomposition [20], Cayley transform [21], square decomposition [22], Householder reflection [23], and optimization over Stiefel manifolds [24]. These methods decompose the unitary matrix into smaller parameter spaces with mathematical processes that guarantee unitarity; however, the weight matrices in these approaches must be square. For convolutional neural nets with rectangular weights, there are approximation techniques based on least square fitting [25], singular value decomposition [26], and soft regularization [27]. These techniques find the best approximates of the unitary weights, but they do not guarantee the weight matrices are strictly unitary. On the contrary, our approach combines the best of the two schools—it is both strictly unitary and applicable to non-square matrices. Our published work [1] presented in this chapter is the first report on applying the unitary weights based on the Lie algebra framework for a deep convolutional neural network with a comprehensive performance study, aiming to make the unitary network an attractive alternative to conventional normalization methods in inference-time-critical applications.

2.2. Theory

A. Matrix Representation of the Unitary Group

In this section, we explain the mathematical framework [19] for representing the unitary group with unitary matrices, collectively known as the Lie group [28]. Linearization of the Lie group about its identity generates a new set of operators; these new operators form a Lie algebra. Lie algebra is parameterized by the *Lie parameters*, which we arrange as a traceless lower triangular matrix, $\boldsymbol{L}$. We name it Lie parameters because it contains independent trainable parameters for the neural networks.[4] For example, we draw $\boldsymbol{L}$ as a 12 x 12 matrix on the right side of Fig. 2 and highlight the tunable parameters in red. 12 is the dimensionality of the matrix representation, and in general, we denote it as n. The representable algebra through this parameterization is only a subspace of unitary groups. Regardless of the expressivity of the parameterization, the resulting unitary matrix will always preserve the norm; thus, it will guarantee signal stability in deep neural networks. The drawback of using a less expressive parameterization is that it may lead to a lower prediction accuracy for the network, which we will quantify empirically in the result section (Sect. 2.4). In the next few paragraphs, we will walk through the construction of unitary matrices from the Lie parameters ($\boldsymbol{L}$); this process is illustrated in Fig. 2 from right to left.

[4] We assume all matrices are real because the weights and activations in deep learning models are real numbers.

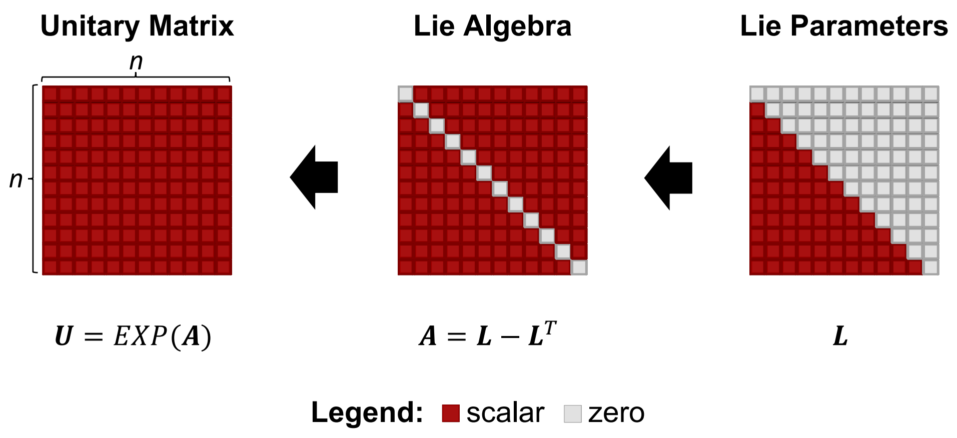

Fig. 2. Unitary matrix construction.[5] There are many ways to construct unitary matrices. In this chapter, we utilize the fact that unitary matrices are matrix presentations of Lie groups, which can be obtained by exponentiating the Lie algebra. From the right, we first create the lower triangular Lie parameters matrix, $\boldsymbol{L}$, for recording the tunable parameters. As shown in the figure, $\boldsymbol{L}$ has $n\ (n-1)\ /\ 2$ trainable parameters. We first initialize $\boldsymbol{L}$ with random numbers and later train them during neural network training using backpropagation. Then, we construct the Lie algebra, $\boldsymbol{A}$, by the equation $\boldsymbol{L} - \boldsymbol{L}^T$ as illustrated in the middle of this figure, assuring $\boldsymbol{A}$ to be anti-symmetric by construction. Lastly, on the left, we exponentiate the Lie algebra ($\boldsymbol{A}$) to obtain the unitary matrix ($\boldsymbol{U}$), which is a matrix representation of the Lie group. Unitary matrices are square by definition. n is the dimensionality of the representation. In this example, n is 12.

[5] Figure modified from our published work [1].

The Lie parameters ($\boldsymbol{L}$) are related to the Lie algebra ($\boldsymbol{A}$) by the following equation:

$$\boldsymbol{A} = \boldsymbol{L} - \boldsymbol{L}^T, \tag{3}$$

where superscript T corresponds to taking the matrix transpose. An essential feature of matrix $\boldsymbol{A}$ (Fig. 2, middle) is that it is an anti-symmetric matrix (i.e., $\boldsymbol{A}^T = -\boldsymbol{A}$). Any compact metric-preserving group, including the orthogonal group, has anti-symmetric Lie algebra [24]. Specifically, the following equation proves that the chosen representation for the Lie parameters will produce an anti-symmetric Lie algebra:

$$\boldsymbol{A}^T + \boldsymbol{A} = (\boldsymbol{L} - \boldsymbol{L}^T)^T + \boldsymbol{L} - \boldsymbol{L}^T = 0. \tag{4}$$

Additionally, in the last step of our pipeline to construct unitary matrices, we exponentiate the Lie algebra, $\boldsymbol{A}$, to obtain the group representation, which will be a unitary matrix, $\boldsymbol{U}$, illustrated on the left side of Fig. 2:

$$\boldsymbol{U} = EXP(\boldsymbol{A}) = \sum_{N=0}^{\infty} \boldsymbol{A}^N / N!. \tag{5}$$

We approximate this matrix exponentiation with an 18-term Taylor series in our implementation. Besides eliminating any term beyond the 18th order in Eqn. (5), we efficiently group the computation to avoid redundant multiplications, a standard approach used in many matrix computation software to save time [30], [31].

We will assume the neural network has synaptic weights that are square for now. In this case, we can use the unitary matrices ($\boldsymbol{U}$) to replace the original weights directly, forcing the neural signals to maintain their norms without explicit normalization. All the intermediate steps for constructing $\boldsymbol{U}$ are algebraic functions as shown in Eqns. (3)-(5), allowing us to train the Lie parameters ($\boldsymbol{L}$) using backpropagation [12] and automatic differentiation [32] together with the rest of the neural net in an end-to-end manner. As mentioned previously, researchers have only applied the unitary pipeline to a small recurrent neural network (RNN), which has a single *square* weight matrix for the feedback loop [19]. Nevertheless, the requirement for the weights to be square severely limits the usefulness of this prior framework. Using the Lie algebra formalism to construct unitary weights is an elegant method to regulate signals, and in the following section, we will expand it to non-square weights by using the Stiefel manifold.

B. The Stiefel Manifold

In the section above, the weight matrices must be square (*n* x *n*), forcing the number of neurons for both the input and output of a particular layer to be identical. This requirement cannot be satisfied in most convolutional neural nets (CNN) because the adjacent layers may have different numbers of neurons; hence, there are many non-square weights in state-of-the-art CNNs. To address this incompatibility between the square unitary matrices and the non-square synaptic weights, we utilize a concept called the *Stiefel manifold* [33], which is a subspace of the unitary group of degree *n* and have a rectangular matrix representation. Fig. 3 is an example process to construct a 12 x 4 matrix in the Stiefel manifold, showing the connection between unitary groups and Stiefel manifolds as follows: The matrix representation of the unitary group is a unitary matrix $\boldsymbol{U}$ with *n* orthogonal columns; $\boldsymbol{U}$ is a square *n* x *n* matrix. By taking the first *k* columns of $\boldsymbol{U}$ (Fig. 3, middle left), we create the matrix representation ($\boldsymbol{V}$) for the Stiefel manifold:

$$\boldsymbol{V} = [\,\bar{u}_1 \quad \bar{u}_2 \quad \dots \quad \bar{u}_k\,] \in \mathbb{R}^{n\times k}, \tag{6}$$

where $\bar{u}_1 \dots \bar{u}_k$ are the orthogonal columns vectors of *n* dimensions copied from $\boldsymbol{U}$. With this construction, $n \geq k$ because we can take *n* columns from $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ at most. The resulting $\boldsymbol{V}$ (Fig.

3, left) is a tall, rectangular matrix with orthogonal columns. For conciseness, we name $\boldsymbol{V}$ the *Stiefel matrix* because it is the matrix representation of an element in the Stiefel manifold; similarly, a unitary matrix is the matrix representation for the Lie group (or the unitary group).

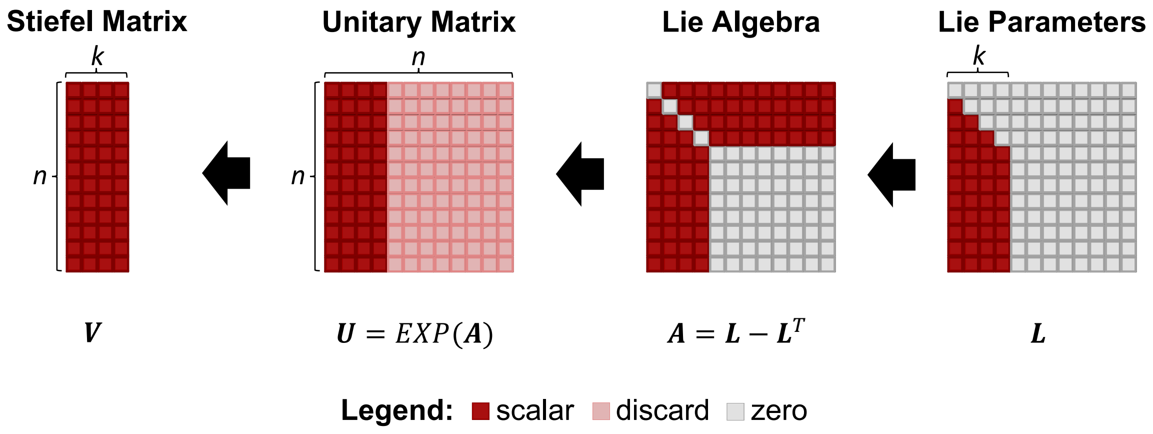
Stiefel Matrix
k
n
Unitary Matrix
n
n
Lie Algebra
Lie Parameters
k
$\boldsymbol{V}$
$\boldsymbol{U} = EXP(\boldsymbol{A})$
$\boldsymbol{A} = \boldsymbol{L} - \boldsymbol{L}^T$
$\boldsymbol{L}$
Legend: scalar discard zero

Fig. 3. Stiefel matrix construction.[6] The first three steps to constructing a Stiefel matrix are almost identical to the construction of an n x n unitary matrix illustrated in Fig. 2. From right to left, we still construct a lower triangular Lie parameter matrix ($\boldsymbol{L}$), compute the Lie algebra ($\boldsymbol{A}$), and exponentiate $\boldsymbol{A}$ to get the unitary matrix ($\boldsymbol{U}$). However, instead of using all n columns in $\boldsymbol{L}$, we only need the first k columns and set the rest to zero. This reduction of parameters is possible because, in the last step, we only take the first k columns in $\boldsymbol{U}$ to construct the Stiefel matrix as shown on the left of this figure. Therefore, the Lie parameters $\boldsymbol{L}$ only contains n x $k - k$ x $(k + 1) / 2$ tunable parameters (right most, shown with highlighted boxes). The Stiefel matrix $\boldsymbol{V}$ is a tall rectangular matrix $\in\mathbb{R}^{n \text{ x } k}$ with orthogonal columns (left most). In this example, n is 12 and k is 4.

[6] Figure modified from our published work [1].

Moreover, it requires fewer Lie parameters in $\boldsymbol{L}$ to define a Stiefel matrix than to specify a unitary matrix. By comparing the Lie parameters in Fig. 2 and Fig. 3, one can visualize their difference: In Fig. 3 (right), $\boldsymbol{L}$ only has k non-zero columns as opposed to the full n - 1 columns illustrated in Fig. 2 (right). Geometrically speaking, a unitary matrix represents a rotation in the n-dimensional space; its columns form a complete set of orthonormal basis vectors in the rotated coordinate system. We utilize the latter interpretation to paint a geometric relationship between unitary groups and Stiefel manifolds—the unitary matrix consists of all n basis. On the other hand, the Stiefel matrix selects k of the n orthonormal basis vectors in an n-dimensional space, forming a subspace of the unitary group. Hence, we only need k columns of Lie parameters ($\boldsymbol{L}$) to fully specify the k basis vectors required for the Stiefel matrix.

The unitary matrix ($\boldsymbol{U}$) is square, but the Stiefel matrix ($\boldsymbol{V}$) is rectangular. And since both have orthogonal columns, we would like to check whether $\boldsymbol{V}$ shares the same norm-preserving property described in Eqn. (2). The columns of $\boldsymbol{V}$ are orthogonal; thus, by construction, we have:

$$\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I} \in \mathbb{R}^{k\times k}. \tag{7}$$

This is trivial to prove because the dot product between different basis vectors is zero while between the same basis vector is one, and $\boldsymbol{V}$ is a collection of basis vectors. With this insight, we repeat the derivation of norm-preservation for Stiefel matrices:

$$||\boldsymbol{V}\bar{x}||_2 = (\boldsymbol{V}\bar{x})^T(\boldsymbol{V}\bar{x}) = \bar{x}^T\boldsymbol{V}^T\boldsymbol{V}\bar{x} = \bar{x}^T\bar{x} = ||\bar{x}||_2, \tag{8}$$

where $\bar{x} \in \mathbb{R}^k$ is the input vector. $||\cdot||_2$ denotes the Euclidean norm, a distance measure calculated by squaring all the coordinates, summing the results, and taking the square root. When we use $\boldsymbol{V}$ as the weight matrix for connecting two neural layers, we will have the following input-output relationship:

$$\bar{y} = \boldsymbol{V}\bar{x}, \tag{9}$$

where $\bar{x} \in \mathbb{R}^k$ are the activations of k input neurons (a column vector), $\bar{y} \in \mathbb{R}^n$ are the activations of n output neurons (also a column vector), and $\boldsymbol{V} \in \mathbb{R}^{n \times k}$ is the Stiefel matrix with *k* orthogonal column vectors of dimension *n*. Putting the previous two equations together, we prove that the norm of the activation signal is preserved between two adjacent neural layers with different sizes:

$$\|\bar{y}\|_2 = \|\boldsymbol{V}\bar{x}\|_2 = \|\bar{x}\|_2. \tag{10}$$

To visualize this norm-preserving dimensionality expansion process, we picture a standard basis, ***I***, as the coordinate system of an *n*-dimensional vector space, where ***I*** is the *n* x *n* identity matrix. ***U*** defines a rotated basis in the same vector space, and ***V*** selects *k* of these basis vectors to form a subspace. We can interpret the input vector ($\bar{x}$) as coordinates with respect to the basis set ***V*** (i.e., each of the *k* scalars in $\bar{x}$ is a coefficient for a corresponding basis vector in ***V***). When we compute $\boldsymbol{V}\bar{x}$ in Eqn. (9), we are mixing the set of *k* basis vectors according to the ratio prescribed by $\bar{x}$, resulting in an *n*-dimensional output vector $\bar{y}$. For geometric intuition, we explain Eqn. (9) as a change of basis from the *k*-dimensional subspace ***V*** to *n* dimensions. Since ***U*** is an orthogonal basis, we can expand $\bar{x}$ from the *k*-dimensional ***V*** subspace to the *n*-dimensional ***U*** coordinate system by simply setting zeros for the extra *n* – *k* coordinates without modifying the vector norm. Finally, reading out the resulting coordinates in the standard basis (***I***) is the same as performing a unitary rotation (***U***), which preserves the vector norm. Thus, in addition to the mathematical proof in Eqns. (7)-(10), the norm-preserving property of the Stiefel matrix can be explained geometrically in the case of increasing numbers of neurons (i.e., *n* ≥ *k*).

A key limitation of the Stiefel matrix is that it does not work with decreasing numbers of neurons between two adjacent neural layers. Previously, we construct the Stiefel matrix ($\boldsymbol{V}$) by taking k columns from a n x n unitary matrix in Eqn. (6), implicitly creating a constrain that k has to be less or equal to n. When $\boldsymbol{V}$ is used as the synaptic weight, k corresponds to the number of neurons in the previous layer and n is the number for the next layer as shown in Eqn. (9). By definition, $\boldsymbol{V}$ is a tall, rectangular matrix (i.e., $n \geq k$), forcing the neural layers to expand. Furthermore, we cannot just transpose to $\boldsymbol{V}$ to resolve this limitation. $\boldsymbol{V}^T\bar{x}$ computes the dot products between $\bar{x}$ and the basis vectors in $\boldsymbol{V}$, measuring how much it aligns with each basis vector. According to the Pythagorean theorem, this projection results in a shorter vector compared to $\bar{x}$ because we dispose of those vector components outside of the $\boldsymbol{V}$ subspace (we take only k basis from $\boldsymbol{U}$ to build $\boldsymbol{V}$).

In practice, there are multiple ways to solve this problem. The easiest solution is to normalize the output to recuperate the signals lost in missing dimensions (only for those layers with a shrinking number of neurons). We decide to use this approach in this dissertation because normalization is only required for a small portion of our network, and the normalization we use merely divides the output vector with its Euclidean norm without any statistical adjustments. Even though it is not ideal to add normalization back to portions of our network, the unitary weights offer other benefits over conventional normalization. Researchers have found orthogonal weights lead to more efficient filters with fewer redundancies [27]. Another solution is to concatenate multiple unitary matrices horizontally to build a wide weight matrix, which preserves the norm for segments of the input vector $\bar{x}$. Lastly, it is also possible to minimize situations requiring wide weight matrices by carefully designing the mapping between the weight matrices and the convolution filters.

C. From the Stiefel Matrix to Convolution Filters

Convolutional layers have weight matrices commonly referred to as filters that convolve with the input image [12]. The input-output relationship for a convolution layer is purely algebraic:

$$\boldsymbol{Y}(i,j,k) = \sum_m \sum_l \boldsymbol{F}(m,l,k)\,\boldsymbol{X}(i+m, j+l), \tag{11}$$

where $\boldsymbol{Y}$ is the output activations, $\boldsymbol{X}$ is the input greyscale image, $\boldsymbol{F}$ is the convolution filters, m and l are dummy indices for convolution, i is the index for the horizontal direction, j is for the vertical direction, and k is the channel direction that corresponds with the filter dimension. In Fig. 4 (bottom), we illustrate an example convolution layer with twelve 2 x 2 filters ($\boldsymbol{F}$), an 5 x 5 greyscale image ($\boldsymbol{X}$), and an output tensor of size 4 x 4 x 12 ($\boldsymbol{Y}$), assuming padding of zero and stride of one for the convolution. The last dimension of $\boldsymbol{Y}$ is the output channel; it always matches the number of convolution filters we have. In our notation, $\boldsymbol{F}$ concatenates all 12 convolution filters in the last dimension; hence, its dimensionality is 2 x 2 x 12.

Moreover, we can succinctly represent the convolution as a single matrix multiplication through the Toeplitz matrix arrangement [27], [34], [35]. Suppose we arrange the input image ($\boldsymbol{X}$) as a Toeplitz matrix ($\boldsymbol{X}_{toeplitz}$) and reshape the convolution filters ($\boldsymbol{F}$) to a 2-dimensional matrix ($\boldsymbol{V}$) as illustrated in Fig. 4. In that case, the convolution simplifies to a matrix multiplication between $\boldsymbol{V}$ and $\boldsymbol{X}_{toeplitz}$ as shown in Fig. 4 (top). $\boldsymbol{V}$ is the Stiefel matrix from Fig. 3, stabilizing the activations across a neural layer with its norm-preserving property as proven in Eqns. (7)-(10).[7] Effectively, we convert the convolution between high-dimensional tensors to a *multiplication*

[7] In which $\bar{x}$ is a column of $\boldsymbol{X}_{toeplitz}$ and $\bar{y}$ is a column of $\boldsymbol{Y}_{2d}$ in Fig. 4 (top).

between 2-dimensional matrices. The mapping between $\boldsymbol{V}$ and $\boldsymbol{F}$ requires careful consideration as it will affect along which dimension the neural signal is normalized.

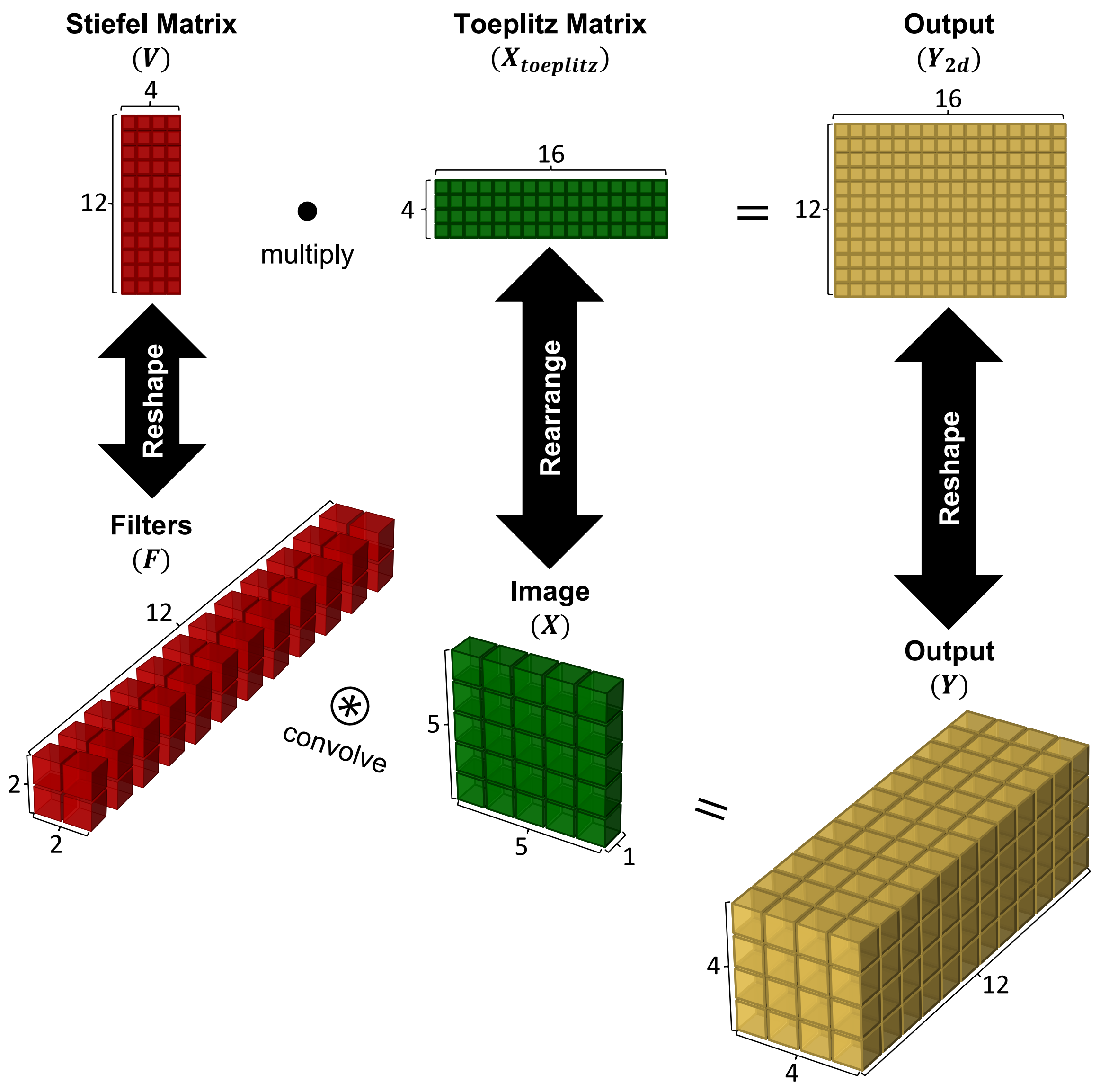

Fig. 4. The Stiefel matrix as convolution filters. We reshape the rows of the Stiefel matrix ($\boldsymbol{V}$) into 2 x 2 convolution filters to ensure signal stability (left). The filters convolve with the input image ($\boldsymbol{X}$), which has 5 x 5 greyscaled pixels (middle). $\boldsymbol{X}$ is rearranged as a Toeplitz matrix ($\boldsymbol{X_{toeplitz}}$) such that the convolution between $\boldsymbol{F}$ and $\boldsymbol{X}$ becomes matrix multiplication between $\boldsymbol{V}$ and $\boldsymbol{X_{toeplitz}}$ [27]. We reshape the result of the

multiplication (***Y_{2d}***) into a 4 x 4 x 12 output tensor (***Y***); 12 is the number of output channels (right). Matrix dimensions are for example only; each box represents a scalar.

2.3. Method

A. __Network Architecture__

The objective of our experiment is to show that the neural network becomes faster in making predictions during inference when we remove normalization from the network architecture; moreover, we want to demonstrate that unitarity is necessary to stabilize network training (i.e., removing normalization without adding unitary constraints will make the network fail to converge in training). Our framework for constructing orthogonal convolution filters from the Stiefel matrix (Sect. 2.2) applies to any neural nets, and we decide to test our idea on a particular network architecture named the residual neural network (ResNet), which is the state-of-the-art for image recognition (Fig. 5, middle). ResNet uses residual connections to improve training; more specifically, the residual connection functions as a gradient highway for the error signal to access the earlier layers in the network during backpropagation training [10]. The residual connection is the long, vertical bypass path in Fig. 5.

In addition to having residual connections, ResNet is also a type of convolutional neural network because it has many layers of convolution filters as shown in Fig. 5. Convolution is the best approach for image recognition due to the translational symmetry of the convolution function. This means that if we construct the network using convolution filters, shifting the objects in a picture will not change the prediction output as we will expect. We apply the Stiefel matrices (Sect. 2.2C) to the convolution filters and remove the normalization layers in ResNet (Fig. 5, right), and we name our architecture the unitary residual neural network (UniResNet). Our architecture is a narrower and shorter variant of the popular ResNet-50, which has 50 layers with weights (the number denotes the total layers with tunable synaptic weights). The number of layers is adjustable by simply appending more unit blocks to the network as shown in Fig. 5. A deeper network results in a more complicated mathematical function (equivalent to a higher-order polynomial) and is more susceptible to overfitting. *Overfitting* is a phenomenon in which the network fits the training data perfectly well but fails to generalize to new data. When a network overfits, its training accuracy is high, but its testing accuracy suffers greatly. We picked a smaller model to prevent overfitting because ResNet-50 was designed for the more complex ImageNet dataset that has 1000 object classes as opposed to the CIFAR-10 dataset we use in our experiments. CIFAR-10 has only ten classes (see Sect. 2.3B for details). Our UniResNet-44 architecture convolutional layers with a fully connected layer at the end for projecting the high-dimensional neural signals to ten output classes; we document the sizes and numbers of convolutional filters for each layer in Fig. 5 below (left & right) for data reproducibility. Also, we study the scalability in terms of depth with the 92 and 143-layer networks (i.e., UniResNet-92 and UniResNet-143). Unless specified otherwise, we refer to UniResNet-44 as "UniResNet" in the following sections. See our source code for details.[8]

An important implementation detail is that we cache the Stiefel matrix after training to save inference time (the time to predict the class label of a given input during inference). The steps to construct the Stiefel matrix are explained in Sect. 2.2 and depicted on the top-right corner in Fig. 5 with the label "activated for training." During inference, the resulting Stiefel matrix is retrieved from the cache, so we do not spend time computing the unitary matrix ($\boldsymbol{U}$) again as exponentiating the Lie algebra ($\boldsymbol{A}$) can take significant computing resources. The translation between $\boldsymbol{V}$ and $\boldsymbol{F}$ (Fig. 5, middle-right) is a simple tensor reshaping operation detailed previously in Sect. 2.2C.

[8] https://github.com/h-chang/uResNet

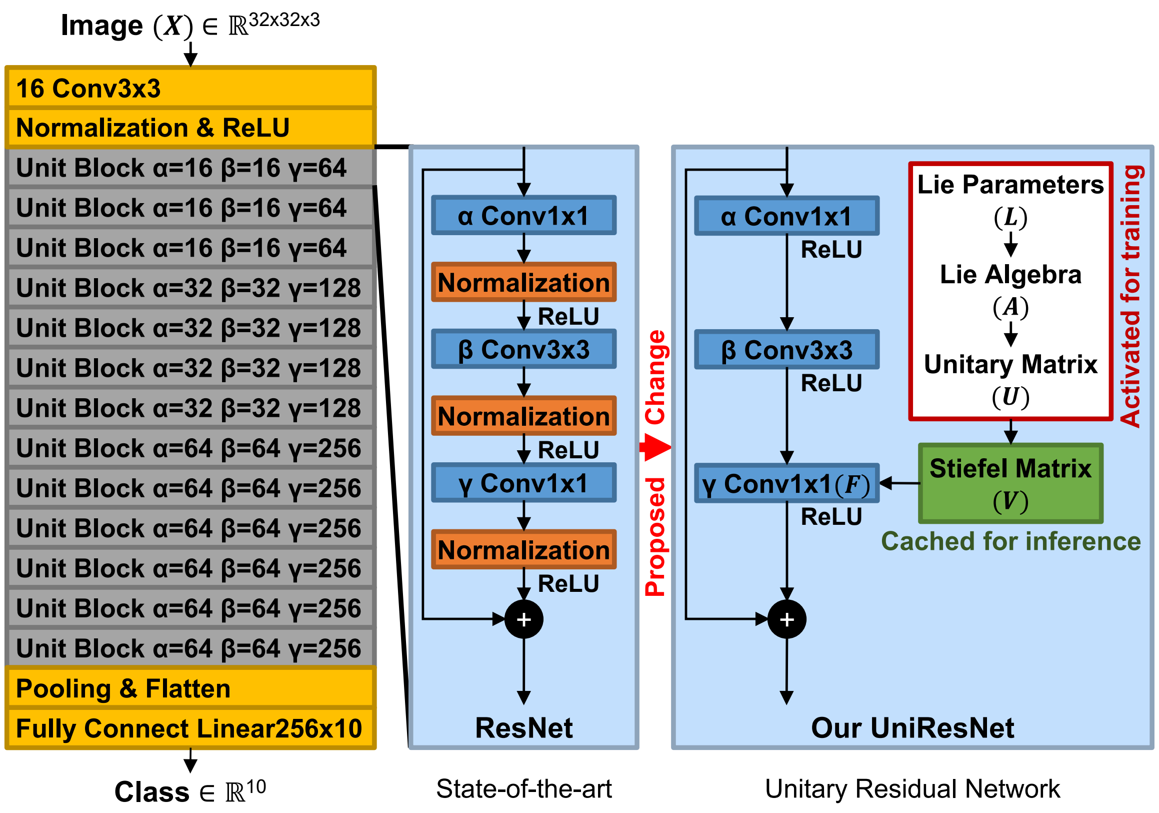

Image $(X) \in \mathbb{R}^{32x32x3}$
16 Conv3x3
Normalization & ReLU
Unit Block α=16 β=16 γ=64
Unit Block α=16 β=16 γ=64
Unit Block α=16 β=16 γ=64
Unit Block α=32 β=32 γ=128
Unit Block α=32 β=32 γ=128
Unit Block α=32 β=32 γ=128
Unit Block α=32 β=32 γ=128
Unit Block α=64 β=64 γ=256
Unit Block α=64 β=64 γ=256
Unit Block α=64 β=64 γ=256
Unit Block α=64 β=64 γ=256
Unit Block α=64 β=64 γ=256
Unit Block α=64 β=64 γ=256
Pooling & Flatten
Fully Connect Linear256x10
Class $\in \mathbb{R}^{10}$
α Conv1x1
Normalization
ReLU
β Conv3x3
Normalization
ReLU
γ Conv1x1
Normalization
ReLU
+
ResNet
State-of-the-art
Proposed Change
α Conv1x1
ReLU
β Conv3x3
ReLU
γ Conv1x1(F)
ReLU
+
Lie Parameters (L)
Lie Algebra (A)
Unitary Matrix (U)
Activated for training
Stiefel Matrix (V)
Cached for inference
Our UniResNet
Unitary Residual Network

Fig. 5. Our unitary residual neural network (UniResNet) architecture.[9] Our UniResNet (right) speeds up inference significantly by removing normalization in the state-of-the-art ResNet (middle). To maintain signal stability in our UniResNet, we use the Stiefel matrix (V) to ensure that the norm of the input vector is preserved (see Sect. 2.2B). All neural layers in the network are listed on the left; we enlarge one of the unit blocks to illustrate its contents. Each unit block contains three convolution layers with different numbers of convolution filters. We label the number of filters with α, β, and γ in the figure. For instance, β Conv3x3 for a layer with $\beta = 16$ means that there are 16 convolution filters of size 3 x 3 in that layer. The rectified linear unit (ReLU) is used as nonlinearity at locations indicated in the figure.

[9] Figure modified from our published work [1].

B. ***Datasets Characteristics***

We use the CIFAR-10 image recognition dataset created by the Canadian Institute for Advanced Research; it contains 60,000 32 x 32 color images with ten labeled classes and is freely available for download [36]. The ten classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks with 6,000 pictures for each class, and the recognition task is to predict the class of a given image. We split the dataset into 50,000 training and 10,000 test images with the same data argumentation scheme as the original ResNet-50 paper [10]. Following the convention used in machine learning, we call the 50,000 training data the "training set" and the 10,000 testing data the "test set." Additionally, we also study the unitary neural network's susceptibility to overfitting with the CIFAR-100 dataset, which contains 100 different types of objects [36].

C. ***Training Details***

We modified the source code found in this reference [37] for comparison against conventional normalization techniques, sharing the same learning rate (0.1), learning schedule (that divides the learning rate by 10 at 100, 150, and 200 epochs), batch size (128), and training epochs (250). The only modification we made for the unitary neural net is that we added the unitary pipeline using the method described previously in Sect. 2.3A for the convolutional layers. We also removed all the normalization blocks in the unitary version. We trained the regular and the unitary networks with the stochastic gradient descent optimizer in PyTorch with a momentum setting of 0.9 and weight decay of 0.0002. We measured the neural networks' speed and memory usage by simulating each neural architecture one at a time on a single NVIDIA RTX3090 graphics card with 24 GB of total video memory.

D. ***<u>Caching the Unitary Weights</u>***

During training, the entire neural pathway is enabled, including the block that contains the Lie parameters, Lie algebra, and Lie group (Fig. 5, top-right). Gradients are backpropagated from the output to update the Lie parameters. After training is complete, the best unitary weights are cached; thus, we do not need to recompute the unitary weights during inference.

2.4. Results & Discussion

With our proposed unitary convolutional neural network from Sect. 2.3, we compare the performance of our proposal against popular normalization methods and summarize the main results of our experiment in Fig. 6 below. By removing the network's unitary pipeline (the block with Lie parameters, Lie algebra, and Lie group in Fig. 5) during test time, we achieved a much faster inference speed than other normalization methods, including the batch norm [17], group norm [38], layer norm [39], and instance norm [40]. Each of these methods addresses a specific problem; therefore, the designer might favor one over the other depending on the application. With our unitary convolution, we offer the community another tool in the toolbox that is lightning fast—32% faster than the instance norm in inference. We compute the speedup by comparing the inference time between the unitary network and the instance norm in Fig. 6(e). Our method shares many characteristics with the instance norm; however, instead of normalizing based on the neural signals' statistics, we devise a set of unitary weights to ensure signals maintain their norm per Toeplitz matrix row. Compared to the instance norm's training time, our training time for the unitary network is also long due to the need to perform matrix exponentiation. Still, it is possible to further expedite it by limiting the frequency that we exponentiate (i.e., sharing the same unitary weights for several iterations) or utilize iterative methods to implement the unitary constraint. The result shown in Fig. 6 is measured without weight sharing during training.

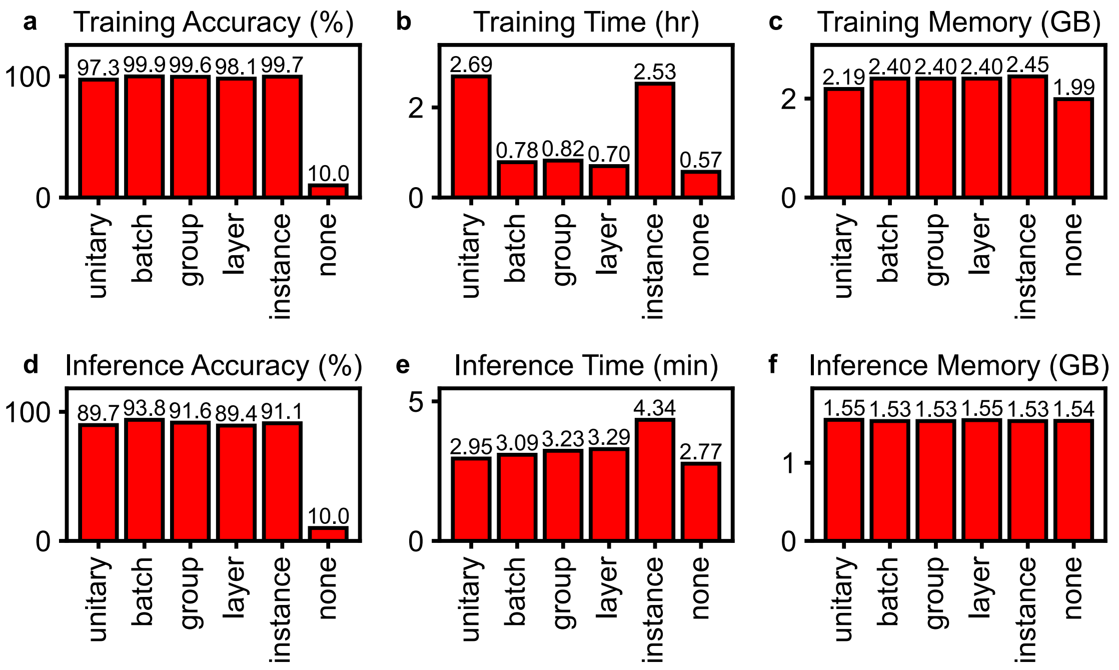

Fig. 6. Performance of our UniResNet vs. other normalization methods.[10] Other methods include the batch norm [17], group norm [38], layer norm [39], and instance norm [40]. We also included the case without any normalization for comparison (none). We used the residual network (ResNet) architecture with 43 convolutional layers to measure the accuracy, the time, and the memory benchmarks when the networks perform image recognition tasks on the CIFAR-10 dataset as described in Sect. 2.3A. CIFAR-10 has ten unique classes of objects. **(a, d).** Training and inference accuracies, respectively. The accuracy reports the percentage of time the network determines the image class correctly with one try. **(b, e).** Training and inference time, respectively. The training time is the time to train the network with 12.5 million images, while the inference time reports the time to recognize 2.5 million images. **(c, f).** Training and inference memory,

[10] Figure from our published work [1].

respectively. Because we trained these networks on graphics processors, memory benchmarks measure the maximum video memory a network consumed during each operation mode. All metrics are average measurements over four simulation runs.

In our experiment, both unitary network and batch normalization do not calculate running statistics (i.e., means and variances) during inference while group, layer, and instance norms track running statistics in the test set. Batch normalization is the second fastest and can potentially match the speed of the unitary network if the batch normalization layer is absorbed into the previous convolutional filters. However, batch normalization will not perform well in applications that require small batch sizes or normalization per data sample such as making adjustments to the contrast of individual images [40]. Group, layer, and instance normalizations work on a per-image basis; the difference between them is the number of channels that they average over. In our experiment, we picked a group size of eight; hence, the group normalization needs to keep track of eight means and variances per image. Contrary to the layer norm that only requires one mean and one variance per image, the instance norm tracks as many means and variances as the number of channels, which is up to 256 in our architecture. Our UniResNet maintains the L2 norm of each column in the Toeplitz matrix $\boldsymbol{X}_{toeplitz}$ (Fig. 5) representing the input activations, delivering similar effects as the instance norm but without the inference speed penalty. The mapping between the filters and the unitary matrix determines which dimension of the activation map the unitary network is effectively normalizing.

The unitary network also uses less temporary memory (dynamic random-access memory or DRAM) required to backpropagate neural signals through the normalization layers during training; more specifically, 8% less than all other normalization methods. Despite our advantages in inference speed and training memory, unitary networks' accuracy is slightly lower in general. Unitary weights constrain the signals to be on the *n*-sphere (or *k*-sphere since we have *k* dimensions) by design and are less expressive than free weights. Nevertheless, our accuracy is comparable to other normalizations and even surpasses the inference accuracy of the layer norm. An additional advantage for the unitary network is apparent when we save the model parameters to hard disks: as we demonstrated in Fig. 3, the matrices encoding the Lie parameters have many zeros. This leads to better compression for the parameter files. An approximation for model size saving is roughly a 15% to 50% reduction in disk space when working with unitary convolutional architectures. We compute the 50% reduction by leveraging the fact that we only need to record half of the values in a triangular Lie parameter matrix, assuming that the weight matrix is square as shown in Fig. 2.

We observe that the unitary neural networks are less susceptible to overfitting. Using the CIFAR-100 dataset and the same network structure (UniResNet-44), we discovered that unitary networks have a smaller difference between the training loss (1.44) and the testing loss (1.62). While the conventional ResNet has a larger difference between the training loss (0.07) and the testing loss (1.56). A smaller difference between training and testing loss means that the network generalizes to new data better; in other words, it has less overfitting. Furthermore, our unitary networks can be deepened to 100+ layers without the costly normalization blocks: the 92-layer version (UniResNet-92) achieves 99.6% and 90.4% in training and testing accuracies, respectively, on CIFAR-10. Similarly, the 143-layer version (UniResNet-143) delivers 99.7% and 90.7% in training and testing accuracies.

2.5. Conclusion

We report here the first instance of using unitary matrices constructed according to the Lie algebra for rectangular convolutional filters, which eliminates the exploding and vanishing activations in deep convolutional neural networks. With clear geometrical interpretations, our theory is a breakthrough based on rigorous, exact construction of the unitary weights applicable to all types of neural networks including but not limited to convolution. The key innovation is that we found a way to ensure signal unitarity with unitary weight matrices of any shapes and dimensions such that the neural signals will propagate across the network without amplification or degradation. Moreover, unlike traditional normalization, our approach has the least impact on inference time, achieving a 32% speedup in recognizing color images when compared to instance normalization. The effective normalization dimension is adjustable in our framework through the mapping between the convolutional filters and the unitary matrices. Our proposal also reduces hard disk storage by up to 50% depending on the neural architectures. The presented framework establishes unitary matrices as a design principle for building fundamentally stable neural systems.

Chapter 3 — Unitary Neural Nets For Robust Natural Language Processing

Unitarity improves the robustness under adversarial attacks.

3.1. Introduction

A. Natural Language Processing

The objective of natural language processing (NLP) is to automatically analyze human-generated texts with machines, and the specific NLP task we study in this dissertation is text classification. That is, we aim to design robust machine learning models that can categorize the topic of a paragraph, understand the causal relationship between sentences, and infer the writer's sentiment. To achieve this aim, we start by improving the state-of-the-art neural network in NLP: the Bidirectional Encoder Representations from Transformers (BERT)—a network structure designed specifically for understanding languages.

The neural architecture of BERT [41] is shown in Fig. 7. [42]. Its first neural layer contains three look-up tables for encoding a sentence into a vector, and we call these tables embeddings (i.e., Word, Position, and Token shown in Fig. 7, top left). For instance, in Word embedding, each word in the dictionary is represented by a vector of 768 dimensions. The vectors are first initialized randomly and then trained to represent the relationship between words in the vector space geometrically. This also means that, with successful training, BERT will group similar words closer together in the vector space [43]. Likewise, the Position embedding represents the position of the word in a sentence, and the Token embedding records the token type.[11] At the end of the embedding section (labeled as Embed in Fig. 7), the vectors are summed together and normalized before further processing.

[11] More specifically, the Token embedding is used in a next-sentence-prediction task, in which we mark whether a word belongs to the previous sentence or the next sentence. It is unimportant for our discussion.

After converting words to vectors, BERT analyzes the sentence embedding by using an attention mechanism similar to index cards in a library back in the day: The Query (Fig. 7, top-right) is a linear transformation for devising a question to ask, the Key is the indexes for the relevant information, and the Value is the information itself. Each of these layers has a unique weight matrix for the linear transformation. For simplicity, we call the resulting neural activations query, key, and value, respectively. The query and key are multiplied together to compute an attention map, highlighting the location BERT should pay attention to in the value. The attention map multiplies with the value to retrieve the actual information. The resulting activations pass through multiple linear transformations (Dense, Expand, and Reduce in Fig. 7, middle-right) and normalizations. Bypass connections are provided as a gradient highway to improve training convergence like in the residual neural networks discussed earlier in Sect. 2.3A.

The attention layer is repeated 12 times to transform the sentence embedding into a succinct neural representation for the classifier, a linear transform. Finally, we use a projection matrix for reducing the dimensionality of the number of classes for a classification problem (Fig. 7, bottom-left). The final activation values ($\bar{y}$) are often referred to as "*logits*" in the literature.[12] During inference, we use the argmax function to identify the class that has the highest logit, resulting in an integer class label. Also, the *argmax* function takes in a vector and returns the index of the largest vector element. For training, the logits are normalized with softmax before passing to the cross-entropy loss (Fig. 7, bottom-right). We will explain softmax and cross-entropy in the following paragraph.

[12] The name logits comes from logistic units in mathematics; they are the inputs to a partition function for computing propabilities in logistic regression, representing a unit change of probability in the log scale.

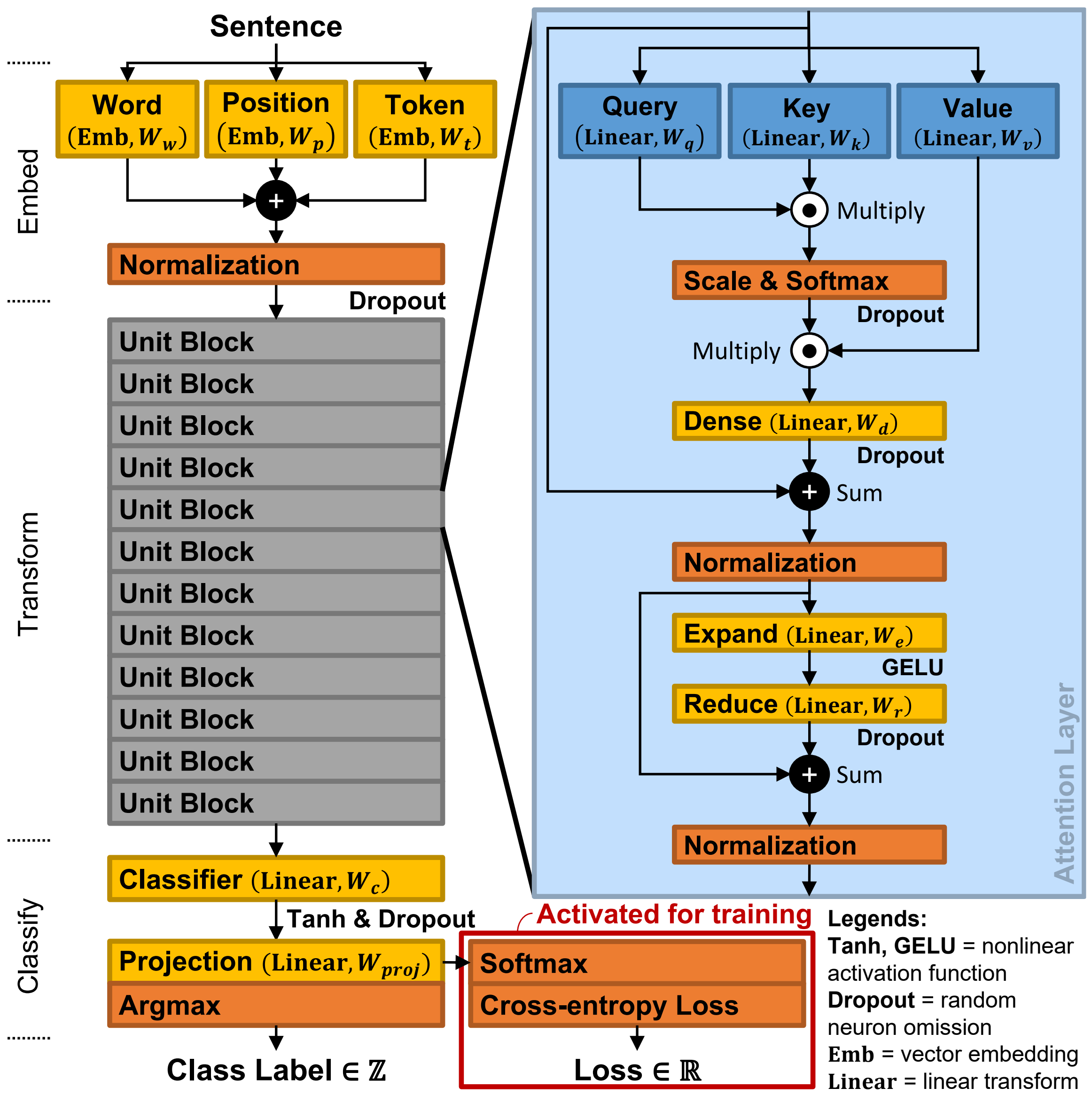

Fig. 7. Bidirectional Encoder Representations from Transformers (BERT) neural architecture. BERT [41] represents a sentence in digital computers by encoding its words using vector embeddings. For each word, we look up three vectors, representing its meaning, position, and type using the Word, Position, and Token embedding modules, respectively (top-left). The network then *transforms* the sentence vector using 12 attention layers (middle-left). The details of an attention layer are illustrated on the right. After the series of transformations, the resulting encoding represents the meaning of the

sentence succinctly; then, we feed this final encoding to a classifier for categorizing the sentence into one of the predefined classes. As for notations, $(\mathbf{Emb},\ \boldsymbol{W_w})$ indicates that the type of neural module is an embedding layer with weight matrix $\boldsymbol{W_w}$; $(\mathbf{Linear},\ \boldsymbol{W_q})$ denotes that the layer is a linear transform with weight matrix $\boldsymbol{W_q}$. Tanh is the hyperbolic tangent; GELU is the Gaussian Error Linear Unit [44]. They are nonlinearities similar to the ReLU.

The *softmax* is a partition function for renormalizing each logit into the probability of the data belonging to a particular class (p_i). Amplifying the logit that has the highest value and suppresses the rest, it is named softmax because is a smooth, differentiable function in contrast to argmax, which simply picks the maximum element. The softmax is defined as:

$$p_i = \frac{e^{y_i}}{\sum_{i=0}^{n_c-1} e^{y_i}} \forall\, i, \tag{12}$$

Where n_c is the number of classes in the classification problem and y_i is a vector element of the logits $\bar{y}$. p_i represents the probability of the datum belonging to class *i*. We then collect all p_i in a vector and denote it as $\bar{p}$ (i.e., $\bar{p} \triangleq [p_0, p_1, \dots, p_i]^T$).

The *cross-entropy loss* is a distance measure for training the neural network. A neural network is a nonlinear function *f(x)* with tunable parameters $\boldsymbol{W}$ for mapping an input $\bar{x}$ to an output $\bar{y}$ (i.e., $\boldsymbol{W}$ is a high-dimensional tensor that records the weights for all neural layers in the network). When the softmax function in Eqn. (12) is used to renormalize $\bar{y}$, the final output of the network becomes $\bar{p}$. To backpropagate the error for training $\boldsymbol{W}$, we not only need a set of data points to exemplify the desired input-output relationship but also a *loss function* that measures the distance between the current network output ($\bar{p}$) and the desired output (*LABEL*).

The pair ($\bar{x}$, $LABEL$) exemplifies a datum in the dataset, relating a feature ($\bar{x}$) with a class label annotated by humans. In our notation, $\bar{p}$ and $LABEL$ do not have the same dimensionality. More specifically, $\bar{p}$ is a vector of dimension n_c, and $LABEL$ is an integer ranging from 0 to $n_c - 1$. Often, the loss function is also called the error function or the objective function in the literature, and the most common loss function used for BERT is the cross-entropy loss. The cross-entropy loss is defined as the following for a datum:

$$cross\ entropy\ loss \triangleq -\sum_{\substack{i=0 \\ i \neq LABEL}}^{n_c-1} log(p_i), \tag{13}$$

where $p_i \in \mathbb{R}$ is the probability of the sample being in class i as defined in Eqn. (12), n_c is the total number of classes, and $LABEL \in \mathbb{Z} \cap \{0, \ldots, n_c - 1\}$ is the correct class label created by humans. $LABEL$ is an integer. All variables in (13) are scalers, and we average the cross-entropy loss across all data samples in a batch of training data.

BERT has many advantages over its predecessors such as the convolutional neural networks (CNN) and the recurrent neural networks (RNN). Mainly, BERT learns from a massive, unlabeled corpus first and then focuses on a specific category using a small, labeled dataset with little additional training. In machine learning terminology, training for BERT is separated into two phrases: pretraining and finetuning. During the pretraining phase, the network is trained as a masked language model, in which words are randomly masked out and the network is asked to predict the missing words [45]. Pretraining teaches BERT the basic mechanics of the language such as grammar and semantics. Because the dataset does not have human-generated labels, the pretraining phase is considered unsupervised learning. On the other hand, finetuning trains BERT to perform a specific NLP task (e.g., identifying the sentiment of a message as labeled by humans). The finetuning phase is considered supervised learning because the network is trained using a human-labeled data set, and finetuning is much shorter than pretraining. A well-pretrained BERT can solve many downstream tasks with little finetuning.

B. Adversarial Attacks

We define adversarial attacks as follows: an algorithm maliciously injects small perturbations to a neural network to alter its output prediction. In computer vision, a small noise is added to an image to trick the network into believing that the image belongs to a different class [46]. Deep neural networks are sensitive to small noises because they make predictions by drawing polytope decision boundaries in high-dimensional space [11], and they often create extraneous decision regions where no training or test data resides [47]. The attackers aim to create enough perturbation without appearing suspicious to humans but shift the neural response across the decision boundary [48]. *Decision boundaries* are the demarcation of data distributions for different classes in the activation space.

Unlike image processing, NLP works with discrete tokens instead of continuous signals. Therefore, adversarial attacks are contrived differently to attack NLP systems. Specifically for attacks on BERT, by swapping words with their synonyms in a sentence, we maintain the semantics of the sentence while introducing a small change in the vector embedding. Because BERT is a deep neural network with many layers, these small perturbations can be amplified to larger differences as neural signals propagate through the network if the weights are non-unitary. Swapping of words with their synonyms may lead to misclassification in BERT, and we refer to this type of attack as synonym-based attacks [49]–[51]. Another type of adversarial attack is by introducing typographical errors in the sentences [52], [53]. In this case, although the injected typos appear benevolent to the readers, they can manipulate BERT to produce the wrong results because the small perturbations can amplify with non-unitary weights. Sect. 3.3E below lists the BERT NLP attacks we evaluate in this study.

Traditional machine learning for NLP is less susceptible to adversarial attacks because they are often based on rules or statistics with only a handful of processing steps. Some example frameworks include grammar-based parsing (identifying the sentence semantics by the subject, verb, and object relationship), decision trees (a set of decision rules organized in a tree-like graph), term-frequency inverse-document-frequency analysis (statistically analyzing the frequencies of words to identify the topics of a document), support vector machines (making decisions with linear classifiers), and clustering (grouping similar concepts together). Because they use an explicit, small set of criteria for the machine to make a decision, a programmer can easily validate its decision rules. They are robust and simple but lack the intellectual capacity required to understand complex questions and passages. In contrast, the decision mechanism in deep learning is difficult to validate because we cannot interpret the internal neural signals in a neural network. Consequently, lacking formal verification, deep neural networks are more susceptible to adversarial attacks.

C. *The Problem with Current Defense Methods*

To date, the defense techniques against adversarial attacks on BERT can be grouped into two categories:

1. **Adversarial Training with Data Augmentation.** The first technique is to train the model with additional adversarial and augmented data. The designer of the neural network needs to anticipate the attack recipes that the hacker will use, generate adversarial samples accordingly, and train the network with the generated samples to prevent a future attack. The

drawback of this technique is that it requires the designers to predict the attack methods used by the adversaries to create appropriate coverage in the sample space. Furthermore, training will take much longer if we desire full coverage for all types of adversarial attacks. Data augmentation is commonly used together with adversarial training, which generates additional training samples by swapping words with their synonyms or by interpolating the existing data points. One of the best is to combine adversarial training and data augmentation as seen in defense models such as AMDA [54] and MRAT [55] (see Sect. 3.4B for details). For brevity, we will use the term "adversarial training" to refer to defense models that use adversarial training and data augmentation together to achieve state-of-the-art robustness.

2. **Regularization.** The second type of defense against adversarial attacks is by adding a regularization term in the loss function to reduce model complexity and therefore mitigate overfitting. Using polynomial regression as an analogy, we explain overfitting as an undesirable phenomenon in which a machine learning model has too many higher-order terms. The resulting polynomial function is very complex, capable of fitting the training data very well with high training accuracy. However, because of its higher-order terms, it may create curves that are not related to the underlying physics of the variables under study—a model with great capacity learns all the artifacts in the dataset, resulting in poor generalization when the model encounters new data points that it has not seen before. To an extent, neural networks (such as BERT) find the mathematical relationship between the input and the output similar to polynomial curve-fitting, and hence they are suspectable of overfitting the dataset if the network gets too deep. A way to alleviate overfitting is by adding additional constraints on BERT to make the model less expressive. For example, InfoBERT is a robust model that uses an information bottleneck as a regularizer to limit

model complexity [56]. Disadvantages of regularization-based defense include high computational costs and added complexity.

We will demonstrate that our proposed model has superior robustness compared to the state-of-the-art defense techniques based on adversarial training and regularization in Sect. 3.4.

D. Our Proposed Solution

Our unitary multi-margin BERT architecture (UniBERT) is a ubiquitous, simple defense mechanism, and we will outline our UniBERT architecture in Sect. 3.3 below. In the following sections, we will demonstrate that, unlike adversarial training, our model does not depend on the type of attacks used by the adversaries. Moreover, our UniBERT is much more straightforward to implement compared to the regularization methods.

By switching the cross-entropy loss with the multi-margin loss (detailed later in Sect. 3.2A) during finetuning, UniBERT forces the neural representation of different classes to be more distinct and hence increases the decision margin as to be explained next. In this dissertation, we define *decision margin* as the distance between the logits and the decision boundary and *neural representation* as the activation map used by a neural net to encode a concept. In detail, each neuron outputs a scalar value called the activation. Collecting all activations in a layer of neurons results in an activation map. A neural net often has similar activation maps for closely related concepts; hence, we refer to the particular activation map (i.e., neural firing patterns) as the neural representation of a concept. In our UniBERT, the multi-margin loss not only functions as a distance measure to quantify the error between the network output and the desired output but also provides a way to increase the margins (i.e., distances) between the neural representations for different concepts, making the network more robust. Additionally, our UniBERT uses unitary matrices for the synaptic weights to confine small perturbations to achieve an extra boost in robustness by preventing small noise from amplifying across the network (see Sect. 3.2B below). Unitarity maintains the cosine distance between two vectors, hence, constraining the injected perturbations. This counteracts the attacks devised by the adversaries (often in a form of a small change in the input sentence) and stabilizes the final prediction.

As a brief overview for this chapter with detailed explanations to be followed, our scientific discoveries and contributions to the machine learning literature include:

1. The multi-margin loss penalizes different classes from having similar neural representations, providing a cushion to absorb small input perturbations without changing the final prediction.

2. When used together with the multi-margin loss, the unitary weights further boost the adversarial robustness by constraining the injected perturbations across the network.

3. Our UniBERT is attack-agnostic, simple, and unique. It achieves a significant improvement in post-attack accuracy compared to the state-of-the-art defense models.

*E. **Literature Review***

The multi-margin loss is widely deployed in support vector machines (SVM) for traditional machine learning [57], and it has also been shown to make image recognition more robust [58]–[61]. However, it is uncertain whether the same conclusion can be drawn for NLP. In NLP, signals are discrete words instead of pixels (with continuous values ranging from 0 to 255) in a given picture. Likewise, orthogonal constraints have been shown to avert adversarial attacks in computer vision [27], [62]–[64], but orthogonality has not been studied in BERT. Our study is the first time when the multi-margin loss and unitarity are used together to improve the adversarial robustness in NLP systems. The improvement is quite dramatic as shown in the following sections.

3.2. Theory

A. Multi-margin Loss Increases Robustness

As a review, the activation value of a neuron is defined as the sum of the products between the inputs and the weights. The collection of activation values in a layer is called an activation map or activations for short, and the activations of the last neural layer are called logits. Furthermore, a loss function is a distance measure that quantifies the deviation between the neural net's output from the desired output, and a neural net's output may or may not be normalized before feeding into the loss function (we will discuss our chosen implementation later). Unlike most loss functions that only quantify the distance, the multi-margin loss provides an additional margin of safety between the logits and the decision boundary to ensure the network can absorb any input perturbation without changing the prediction outcome. In detail, the extra margin safeguards against attackers from injecting small input perturbations to move the logits across the decision boundary and from sabotaging the network's prediction. The multi-margin loss is defined as:

$$multi\ margin\ loss \triangleq \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{n_c-1} max\left(y_{i,j} + \varepsilon - y_{i,correct}, 0\right), \tag{14}$$

where n is the batch size, n_c is the number of classes, $y_{i,correct}$ is the logit of the neuron corresponding to the correct answer, $y_{i,j}$ is the logit for other neurons; ε is the margin parameter, setting a gap between the logits for different classes as discussed below. All variables are scalars. To provide some intuition for the multi-margin loss, we illustrate how it works in UniBERT with a binary sentiment analysis example in Fig. 8 below.

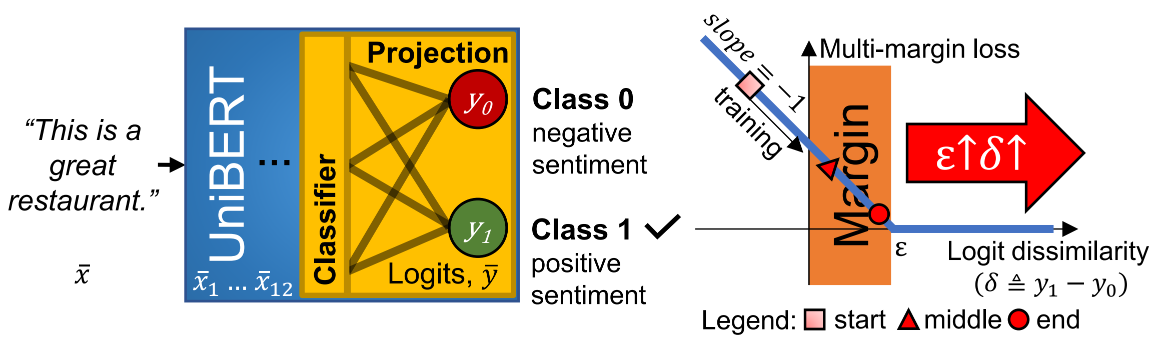
"This is a great restaurant."
$\bar{x}$
UniBERT
$\bar{x}_1 \dots \bar{x}_{12}$
Classifier
Projection
y_0
y_1
Logits, $\bar{y}$
Class 0 negative sentiment
Class 1 ✓ positive sentiment
slope = −1
training
Multi-margin loss
Margin
ε↑δ↑
ε
Logit dissimilarity ($\delta \triangleq y_1 - y_0$)
Legend: ▢ start ▲ middle ● end

Fig. 8. Multi-margin loss for binary sentiment analysis with UniBERT. On the left, UniBERT receives an input sentence $\bar{x}$, transforming it with the embedding and the 12 attention layers (abbreviated as … in the figure) to latent neural representations, $\bar{x}_1 \dots \bar{x}_{12}$. $\bar{x}_{12}$ passes through the classifier and the projection layers to become logits $\bar{y}$, consisting of y_1 and y_0 (scalars) in the binary classification example shown here. If $y_1 > y_0$, the network will predict that the sentence has a positive sentiment (class 1); otherwise, it predicts a negative sentiment (class 0). The multi-margin loss compares $\bar{y}$ with the correct answer labeled by humans (denoted by the checkmark in the figure) and penalizes the network for any insufficient distinction between the two logits (i.e., y_1 - y_0). We name this difference "logit dissimilarity," δ. Additionally, we define a margin parameter ε such that the multi-margin loss is proportional to the lack of the desired margin when $\delta < \varepsilon$ as shown in the figure by the line with a slope of -1. If the network has sufficient margin, the loss is zero (i.e., the flat segment on the right). During training, our UniBERT adjusts its weights from all the layers to minimize loss; training progression is depicted by the square, triangular, and circular markers on the graph. With a larger ε, the multi-margin loss encourages our UniBERT to have highly distinctive logits, making it more difficult for the attackers to move them across the decision boundary with small perturbations and to sabotage the prediction results.

The multi-margin loss is illustrated in the right panel of Fig. 8. For each data sample (indexed by i) in Eqn. (14), the loss is the larger of the two terms between $y_{i,j} + \varepsilon - y_{i,correct}$ and zero, which correspond to the two linear segments of the graph in Fig. 8. The margin parameter (ε) determines the intercept between the two segments. Our objective is to train the network such that the logit for the correct class ($y_{i,correct}$) exceeds the logit of the incorrect class ($y_{i,j}$) by the margin (ε) we specify (i.e., $y_{i,correct} > y_{i,j} + \varepsilon$). If $y_{i,correct}$ fails to meet the target value, $y_{i,j} + \varepsilon$, we will penalize the network with a loss equivalent to the gap to target (i.e., $y_{i,j} + \varepsilon - y_{i,correct}$). This corresponds to the line segment with a slope of -1 (Fig. 8, right panel). On the other hand, if $y_{i,correct}$ already exceeds our target, we set the loss to zero in Eqn. (14), and this is represented by the flat portion in the same graph. As shown later in Sect. 3.4D, a large ε encourages our UniBERT to have distinctive logits for each class and hence increases the decision margin. The separation of neural representations for different classes is critical to prevent the attackers from tampering with the prediction outcome by injecting perturbations to intentionally move logits across the decision boundary. The name multi-margin comes from the fact that there are multiple margins, one for each class.

As for the implementation details, we decide not to use softmax between the logits and the multi-margin loss. The reason is that it dampens the backpropagated gradients as the logits deviate from zero, which slows down learning. Also, in contrast with traditional machine learning, in which features are designed manually, deep learning models such as our UniBERT learn features automatically through backpropagation. This important distinction allows our UniBERT to modify the neural representations in earlier layers (i.e., $\bar{x}_1 \dots \bar{x}_{12}$ in Fig. 8, left) to accommodate a higher margin (ε) and to achieve a lower loss (i.e., the circular marker in Fig. 8, right). Whereas in shallow models such as the support vector machines, training stops as soon as the classifier layer converges without modifying the features (e.g. $\bar{x}_{12}$), leading to a higher loss (i.e., the triangular marker in the same graph). Thus, setting a larger ε is less likely to produce distinctive logits in traditional, shallow models.

B. <u>Unitarity Confines Perturbation</u>

In an adversarial attack, the input perturbation needs to be subtle to avoid detection. A unitary matrix ($\boldsymbol{U}$) maintains the distance between the original ($\bar{x}$) and the perturbed vector ($\bar{x}'$):

$$\|\boldsymbol{U}(\bar{x}-\bar{x}')\|_2 = [\boldsymbol{U}(\bar{x}-\bar{x}')]^T[\boldsymbol{U}(\bar{x}-\bar{x}')] = (\bar{x}-\bar{x}')^T\boldsymbol{U}^T\boldsymbol{U}(\bar{x}-\bar{x}') = \|\bar{x}-\bar{x}'\|_2. \quad (15)$$

This is a corollary of the conservation law for the Euclidean norm of the input vectors as shown in Eqn. (2) of Sect. 1.2. In Eqn. (15), the right-hand side is the norm of the input perturbation, which will need to be small to be discreet. The left-hand side measures the amount of change in the output after the unitary transform. Consequently, unitarity guarantees that small perturbations remain small after the transformation. In its non-unitary counterparts, the linear neural layers may accidentally amplify the perturbations, in which the logits can move across the decision boundary and result in a classification error. Non-unitary weights may suppress the perturbations as well; however, our goal is to eliminate any slight chance of perturbation amplification.

We aim to improve BERT's robustness against adversarial attacks with the unique property of unitary matrices shown in Eqn. (15). More specifically, unitary constraints are enforced in the selected layers in our UniBERT to stabilize the injected perturbations across the network as we will show next. Our UniBERT combines the multi-margin loss in Eqn. (14) and the unitarity in Eqn. (15) to guarantee that there is not only a wide margin to deter the logits from crossing the decision boundary but also a stabilization mechanism to limit the effect of perturbations, delivering the state-of-the-art robustness as demonstrated later in Sect. 3.4 with experimental data.

3.3. Method

A. Network Architecture

Our UniBERT is an enhanced BERT that uses multi-margin loss and unitarity as described above to increase the neural network's prediction accuracy under adversarial attacks. This subsection explains the implementation details for our UniBERT neural architecture, which is illustrated in Fig. 9 below. Compared to the original BERT in Fig. 7, our modifications are:

1. During finetuning, we replace the softmax and cross-entropy loss with just the multi-margin loss on its own (see Fig. 9, bottom). We do not modify the softmax or the cross-entropy loss for pretraining. The reason for this is that multi-margin loss works best for classification tasks; thus, it is applied during finetuning and not during pretraining, which is a masked language modeling task.
2. During both pretraining and finetuning, we force certain layers to have unitary weights, and the selected layers are circled with dash lines in Fig. 9, top-right. The selection of the unitary layers will be explained below. The exact procedure to ensure their weight matrices are unitary is described in the following subsection (Sect. 3.3B).

Although our current implementation has the same number of parameters (110M) as the BERT base, researchers have shown that it is possible to further compress the number of trainable parameters by half [1].

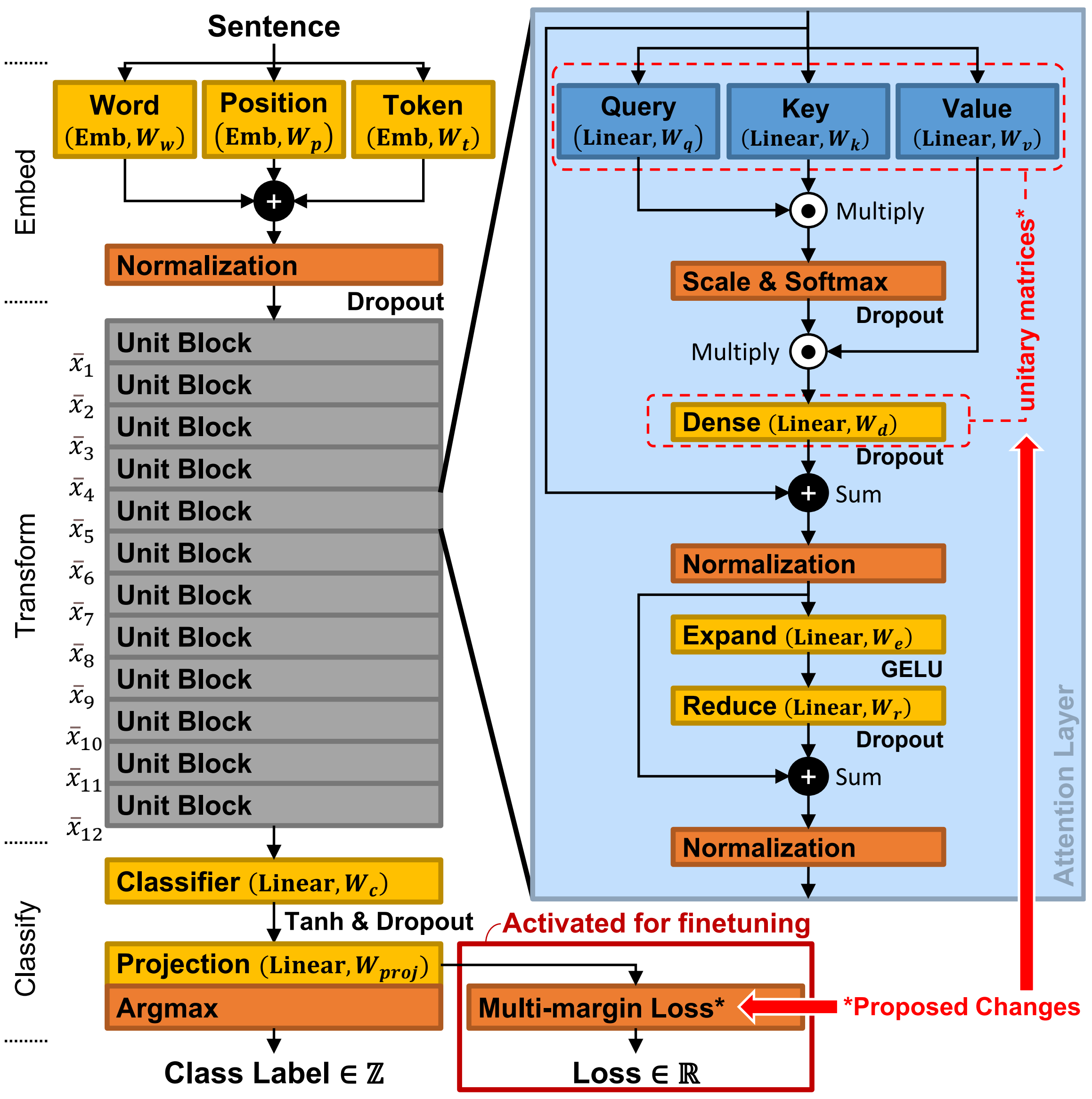

Fig. 9. Our unitary multi-margin BERT (UniBERT) architecture. Like BERT, UniBERT classifies a sentence by first converting words in a sentence vector using the Word, Position, and Token embeddings (top-left). It then transforms this sentence vector using 12 attention layers with details shown on the right of this figure. Our UniBERT is a variant of BERT with the following differences: First, we use the multi-margin loss (bottom) instead of the cross-entropy loss during the finetuning portion of the training process. Second, we enforce unitary constraints on the weights circled with dashed lines

(top-right). These enhancements increase the decision margin and resilience to input perturbations for stabilizing the classification outcomes under adversarial attacks. $\bar{x}_1 \dots \bar{x}_{12}$ denotes the activations after each of the 12 unit blocks, respectively (left).

The unitary layers are selected as follows, and Table 1 below lists all the synaptic weights in UniBERT with their unitarity. Firstly, as mentioned in Chapter 2 previously, only square matrices can be unitary in terms of a strict mathematical definition (Sect. 2.2B); thus, not all layers can be converted to unitary. We decide to apply unitary constraints on all the square weights in UniBERT to maximize the effect of the total unitarity (with one exception to be explained next). Secondly, we keep $\boldsymbol{W_c}$ in the classifier layer non-unitary to increase pre-attack accuracy even though it is also square and thus available for the unitary conversion. $\boldsymbol{W_c}$ is shown near the bottom of Fig. 9 and listed in Table 1. With our unitary convolutional neural net shown before in Sect. 2.4, we learn that in general, unitarity reduces model complexity in exchange for higher neural signal stability. This is the reason why with unitary constraints, pre-attack accuracies decrease (usually by a small amount), and post-attack accuracies increase. As a consequence, we decide to leave the classifier layer non-unitary for a slightly higher pre-attack accuracy.

Our UniBERT is only partially unitary. It has a total of 48 unitary weights and 29 non-unitary weights (Table 1, last two columns). Our selection of unitary layers forces unitary constraints on 62.3% of all weight matrices; i.e., 48 / (48 + 29). Likewise, there are nonlinear activation functions in the network necessary to achieve the required model complexity. The nonlinearities are vital to avoid condensing the transfer function to one linear transformation between the input and the output. While it is impossible to have a purely unitary neural net, our

UniBERT's partial unitarity is sufficient to deter adversarial attacks, and surprisingly, it outperforms many state-of-the-art defense models in post-attack accuracies as we will show later in Sect. 3.4.

Section	**Name**	**Symbol**	**Neural Module**	**Matrix Dimension**	**Unitarity**	**Repeat**
Embed	Word	$\boldsymbol{W_w}$	Embedding	30522 x 768	Non-unitary	1
	Position	$\boldsymbol{W_p}$	Embedding	512 x 768	Non-unitary	1
	Token	$\boldsymbol{W_t}$	Embedding	2 x 768	Non-unitary	1
Transform	*Query*	$\boldsymbol{W_q}$	*Linear*	*768 x 768*	*Unitary*	*12*
	Key	$\boldsymbol{W_k}$	*Linear*	*768 x 768*	*Unitary*	*12*
	Value	$\boldsymbol{W_v}$	*Linear*	*768 x 768*	*Unitary*	*12*
	Dense	$\boldsymbol{W_d}$	*Linear*	*768 x 768*	*Unitary*	*12*
	Expand	$\boldsymbol{W_e}$	Linear	768 x 3072	Non-unitary	12
	Reduce	$\boldsymbol{W_r}$	Linear	3072 x 768	Non-unitary	12
Classify	Classifier	$\boldsymbol{W_c}$	Linear	768 x 768	Non-unitary	1
	Projection	$\boldsymbol{W_{proj}}$	Linear	768 x n_c	Non-unitary	1

Table 1. The complete list of weights used in our UniBERT and their unitarity. Each row in the table is a neural layer with a corresponding weight matrix. In general, unitarity slightly sacrifices the pre-attack prediction accuracy for a large improvement in the post-attack prediction accuracy. We force unitary constraints on all the square weights except the classifier layer (i.e., $\boldsymbol{W_c}$), which we leave unconstrained to achieve higher pre-attack accuracy. Also, non-square weights are difficult to make unitary and therefore left as is. In the projection layer, n_c is the number of classes in the classification task. Italic rows are the unitary weights used in UniBERT; they are non-unitary in the original BERT.

B. Unitary Constraints

The way we convert the non-unitary weights to their closest unitary projections is by QR factorization, a method to decompose any matrix into unitary and non-unitary parts as explained below [65]:

$$\boldsymbol{W} = \boldsymbol{QR}, \tag{16}$$

where $\boldsymbol{W}$ is a non-unitary square matrix, $\boldsymbol{Q}$ is a unitary matrix, and $\boldsymbol{R}$ is an upper triangular matrix. We extract the signs of the diagonal elements in $\boldsymbol{R}$ and construct $\boldsymbol{S}$:

$$\boldsymbol{S} = diag(sign(\boldsymbol{R})) \tag{17}$$

Therefore, $\boldsymbol{S}$ is a diagonal matrix with ±1 on its diagonal. Lastly, we obtain the unitary weights ($\boldsymbol{U}$) by:

$$\boldsymbol{U} = \boldsymbol{QS} \tag{18}$$

$\boldsymbol{U}$ is proven to be unitary by:

$$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{S}^T\boldsymbol{Q}^T\boldsymbol{QS} = \boldsymbol{S}^T\boldsymbol{S} = \boldsymbol{I} \tag{19}$$

After each backpropagation step, we use the procedure detailed in Eqns. (16)-(18) to find the closest unitary projection for selected weights and overwrite them with $\boldsymbol{U}$.

C. Datasets Characteristics

Before we show the training and inferences, we described the key features of the datasets used in pretraining and finetuning to benchmark the differences between BERT and UniBERT. First, we describe the dataset used for pretraining—the Book Corpus (**bookcorpus**), an unlabeled dataset containing 74 million sentences from eleven thousand books [66]. We separate this dataset into two subsets, one for training (95%) and one for testing (5%). The ratio of data in each set is arbitrary. In general, we use the majority of the data for training the neural network but reserve a small portion for validating the accuracy of the trained network. This procedure ensures that the testing accuracy is measured on data that the network has not seen in training for a more accurate estimation of how the network will perform in the real world.

Then for finetuning, we selected three different datasets for a comprehensive evaluation covering multilabel categorization, language inference, and sentiment analysis; respectively, they are listed as follows:

1. AG's News (**ag_news**) is a dataset for news classification, and the goal is to classify the articles into four categories including world news, business news, science & technology, and sports [67].

2. Stanford Natural Language Inference Corpus (**snli**) aims to train machine learning systems that can identify the relationship between a pair of sentences [68]. There are three possible classification outcomes: entailment, contradiction, or neutral.

3. Yelp Reviews Polarity (**yelp**) is a text sentiment analysis dataset constructed by collecting reviews from Yelp.com [67]. The label is either positive or negative.

We highlight the key features of the dataset statistics in Table 2, including the number of classes for the human-generated labels, size of the training data set (training set), size of the testing dataset (test set), and average sample length. and all datasets used in our study follow a uniform distribution in the labels. Sentences are forced to all lower case if the dataset is cased, and all datasets are in English. The datasets are accessible for free from the Hugging Face repository [69].

Dataset	Number of Classes	Training Sample Size	Testing Sample Size	Average Length (Words)
bookcorpus	N/A	70,000,000	4,000,000	13
ag_news	4	120,000	7,600	39
snli	3	550,000	10,000	20
yelp	2	560,000	38,000	136

Table 2. Key features of the dataset statistics for bookcorpus, ag_news, snli, and yelp. bookcorpus is an unsupervised dataset used to train BERT and UniBERT as a mask language model during pretraining. ag_news is a news classification dataset with four classes (world news, business news, science & technology, and sports). snli is a natural language inferencing dataset that asks the model to predict the relationships between two sentences into three categories: entailment, contradiction, or neutral. yelp is a binary sentiment analysis dataset that categorizes user-generated paragraphs into positive reviews or negative reviews. For bookcorpus, each sentence counts as a sample while the other three datasets can have multiple sentences per sample, and the average length column reports the average number of words in a sample. N/A means not applicable.

D. Training Details

Next, we document our training procedure for the proposed UniBERT. We mentioned earlier in the introduction (Sect. 3.1A) that training of BERT-based models has two phases: pre-training and finetuning. For review, pre-training teaches our UniBERT the basic mechanics of the language while finetuning provides on-the-job training for performing a specific task. Each phase requires a slightly different set of hyperparameters because of the differences in the task objectives (masked language modeling vs. classification) and the training datasets.

1. **Pre-training:** We use a fixed language mask with a masking probability of 0.15 and pretrain the UniBERT model from scratch for five epochs with the Adam algorithm on a linear schedule, in which the learning rate will start with 0 at the beginning, linearly rise to 0.0001 (at step 7000), then linearly decay to zero (at the end of the training). Adam is a training algorithm with adaptive moment estimation that alleviates the problem of local minimum by adjusting the momentum of the weight updates [70]. The weight decay reduces the magnitude of the synaptic weights to penalize against extravagant model complexity for better generalization performance. The batch size is 16; the sequence length is 512. These sizes are limited by the memory size of our graphics card, where we run our simulation. Overall, the pretraining procedure takes 5 epochs to converge or equivalently 700,000 steps given our batch size and dataset size. At each step, the unitary weights are first updated using regular gradient descent and then converted to the closet unitary weight using the QR factorization technique explained earlier in Sect. 3.3B.
2. **Finetuning:** We finetune the pretrained models on the three classification datasets: ag_news, snli, and yelp. Similarly, we use QR factorization (Sect. 3.3B) to ensure unitarity on selected weights. Unlike pretraining, the multi-margin loss is used instead with the margin parameter (ε) set to 100 to balance the pre-attack and post-attack prediction accuracies. See Sect. 3.3F for details on how to select the value for the margin parameter (ε). We conduct four independent finetuning runs for scientific rigor; each has a new random initialization on the classifier layer. Finetuning takes five epochs with a learning rate of 0.00005. The batch size is 160 with a sequence length of 128; these settings are limited by the memory of the graphics card used in our simulation.

The training procedures for the other baseline and defense models are left unchanged as described in their original publication. They use neither the unitary weights nor the multi-margin loss during any portion of the training process. We summarize the hyperparameters used for UniBERT training in Table 3.

Category	**Name**	**Pretraining**	**Finetuning**
Model Parameters	Type	Mask Language Model	Classification Model
	Loss	Cross-entropy	**Multi-margin**
	Margin Parameter	N/A	**100**
	Masking Ratio	0.15	N/A
	Sequence Length	512	128
	Unitary Constraint	**Yes**	**Yes**
Training Setup	Dataset	bookcorpus	ag_news, snli, yelp
	Batch Size	16	128
	Epoch	5	5
Backpropagation Algorithm	Optimizer	Adam	Adam
	Scheduler	Linear	Linear
	Learning rate	0.0001	0.00005
	Beta 1	0.9	0.9
	Beta 2	0.999	0.999
	Warmup Steps	7000	500
	Weight Decay	0.01	0.01

Table 3. Training hyperparameter settings for our UniBERT. We first pretrain UniBERT and then finetune it with different datasets and loss functions shown in the table. These hyperparameters are the best-known methods reported in the literature except for the loss, margin, and unitary constraint, which are unique to UniBERT. In detail, the differences to train our UniBERT compared to a regular BERT are: First, UniBERT enforces unitary constraints on the selected layers listed in Table 1 for both pretraining and finetuning. Second, it uses the multi-margin loss with the margin parameter (ε) set to 100 for finetuning. The cross-entropy loss is used for pretraining. The learning rate starts with zero, linearly ramps up to the target learning rate at the

prescribed warmup steps, and linearly ramps down to zeros. The sequence length reports the maximum number of words for each input. N/A means not applicable.

E. Adversarial Attacks

We select three distinct types of attacks for a comprehensive adversarial robustness evaluation:

1. **Textbugger** randomly introduces character insertion, deletion, swap, and substitution to modify BERT's prediction [53]. It is considered a typographic attack.

2. **Textfooler** finds candidate adversarial samples by swapping important words with their synonym; synonyms are found by searching through the neighborhood in the word embedding space using the counter-fitted word embedding [50], [71]. Similar to BERT's word embedding described previously in Sect. 3.1A, the counter-fitted word embeddings are obtained by using a recurrent neural network for converting words to vectors in the embedding space, where semantics are represented geometrically.

3. We add the **Probability Weighted Word Saliency** (or **PWWS**) attack to our evaluation portfolio [51]. It swaps words in a sentence with their synonyms defined in the human-labeled WordNet [72] database. In contrast to Textfooler, which finds synonyms using a distance metric with the word embeddings learned automatically by a neural net, PWWS relies on a thesaurus constructed explicitly by human workers.

Textbugger and Textfooler may not preserve a sentence's meaning; therefore, to create a valid replacement sentence to commence the attack, it needs to check for its semantic similarity using the Universal Sentence Encoder [73], which measures the cosine distance between the original and the perturbed sentences. We use the default threshold settings in the TextAttack [74] framework to reject any adversarial example that changes the meaning of the sentence. In contrast, because PWWS uses the human-labeled WordNet synonym database to generate high fidelity samples as described before; hence, we do not perform additional safeguarding on the generated samples.

A neural network is a machine learning model, and we refer to it as a *model* for short. There are two ways to carry out adversarial attacks on a model: *targeted attacks* vs. *static attacks* [54]. Targeted attacks generate a new set of adversarial samples for each model while static attacks use the same set of adversarial samples to evaluate all neural network architectures (e.g., BERT, UniBERT, InfoBERT…etc.). We use the tougher targeted attacks to evaluate the models' robustness. In detail, we randomly select 1000 data samples from the test set and allow the attack algorithm to make an unlimited number of attempts to the model until it can no longer generate new permutations that meet the similarity criteria or has exhausted all synonyms. The number of attempts the attacker is allowed to make is called the *query budget*. In this dissertation, the *post-attack accuracy* is defined as the ratio of the samples (out of 1000) that survive the series of attacks with an unlimited query budget and still produce the correct classification results, measuring the classification accuracy of the neural net after the attack. Our evaluation methodology is the toughest in the literature, reporting the lowest possible post-attack accuracies. The *pre-attack accuracy* is measured by computing the ratio of the correctly classified sample out of the 1000 test samples without any attack.

F. Selecting The Margin Parameter

Furthermore, there is a new hyperparameter in UniBERT: the margin parameter (ε) for the multi-margin loss in Eqn. (14). We find the best margin parameter, ε, experimentally by sweeping it over five decades between 0.01 and 1000 to simultaneously maximize both the pre-attack and post-attack prediction accuracies of the neural net. We evaluate the tradeoff between pre-attack and post-attack accuracies for the yelp sentiment analysis in Fig. 10, in which our UniBERT is trained to classify the user text into positive or negative reviews. As the margin increases, the post-attack accuracy quickly improves (as shown by the dashed trendline in the figure) while the pre-attack accuracy reduced slightly with ε (as illustrated by the dotted trendline). The best post-attack accuracy happens when ε is 100; therefore, we use this setting for the rest of this dissertation. Depending on the user's tolerance for pre-attack accuracy drop in their applications, they can select an appropriate margin accordingly.

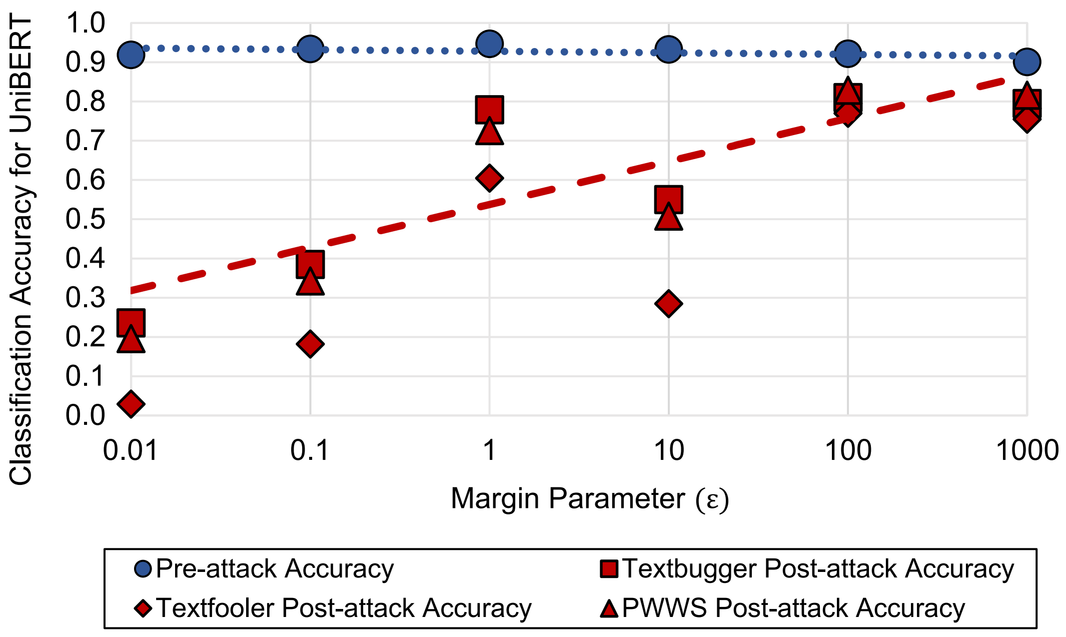

Fig. 10. Classification accuracy vs. margin parameter for the multi-margin loss in UniBERT. To find the most optimal setting for the margin parameter (ε), we use the yelp sentiment analysis dataset to evaluate the pre-attack accuracy of UniBERT in making a correct classification, contrasting it with the post-attack accuracy under various adversarial attacks (namely, Textfooler, Textbugger, and PWWS). As shown in the

figure, a higher ε reduces the pre-attack accuracy (dotted line) while increasing the post-attack accuracy (dashed line) for UniBERT. From this data, we select $\varepsilon = 100$ for the rest of the experiments in this treatise because, with this setting, the UniBERT delivers a large improvement in the post-attack accuracy with a negligible decline in the pre-attack accuracy.

3.4. Results & Discussion

A. Our UniBERT vs. Baseline Models

So far, we have only introduced BERT and UniBERT, but there are many versions of BERT that researchers have developed for various purposes. These neural network models are not specifically designed to defend against adversarial attacks; hence, we call them the baseline models. The goal is to validate the need for a novel defense technique because there would be no reason to modify the existing architectures if the baseline models were already resilient to attacks. We compare our proposed UniBERT against four other versions of BERT, which are listed here:

1. **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**) [41]—This is the original architecture that our work is based on.
2. A **R**obustly **O**ptimized **BERT** Pretraining **A**pproach (**RoBERTa**) [75]—This is BERT with enhanced pretraining to improve its accuracy. We use this model to verify that a better pretraining procedure alone cannot deter adversarial attacks.
3. **A** **L**ite **BERT** (**ALBERT**) [76]—This model reduces the number of parameters up to 18 times by using factorized word embedding and parameter sharing. We select this model to show that model complexity reduction is not enough to prevent adversarial attacks.
4. A **distil**led version of **BERT** (**DistilBERT**) [77]—This model reduces the number of parameters by 40% by using knowledge distillation. Same as with ALBERT, we want to show that reducing the model complexity via transfer learning to a smaller model will not improve robustness.

To conduct a fair comparison for the classification accuracy, we download the pretrained weights for the four baseline models above from the Hugging Face repository [69] and finetune them further using the procedure (see Sect. 3.3D, finetuning) for our selected datasets. Namely, the classification datasets include news categorization (ag_news), natural language inference (snli), and sentiment analysis (yelp), which we have documented their characteristics in Sect. 3.3C above. After finetuning, we first evaluate their pre-attack classification accuracies and then measure their post-attack classification accuracies after performing adversarial attacks on the models. The three attack recipes are the PWWS, Textbugger, and Textfooler adversarial attacks on the textual input, which cover a wide range of typographical and synonym-based NLP attacks. Details of the attack recipes are discussed in Sect. 3.3E before.

The results are given in Table 4, which compares the pre-attack and the post-attack classification accuracy of our proposed UniBERT with the baseline models (i.e., BERT, RoBERTa, ALBERT, and DistilBERT). UniBERT delivers double-digit improvements across all combinations of NLP tasks and attacks over the baseline models. In general, RoBERTa is slightly better than BERT in post-attack accuracy because the extensive pretraining procedure in RoBERTa creates a better language model. On the other hand, both parameter reduction models (ALBERT and DistilBERT) have worse robustness compared to BERT. We conjecture that the lack of model complexity prohibits them to generalize to out-of-distribution samples which happens due to their inferior language models.

Task	Model	Original Accuracy	Post-attack Accuracy		
			PWWS	Textbugger	Textfooler
ag_news	BERT	94.6	32.4	35.8	13.4
	RoBERTa	**95.1**	40.8	48.3	16.7
	ALBERT	92.5	24.1	17.3	8.7
	DistilBERT	93.6	27.9	32.6	10.6
	UniBERT	92.3	**85.4**	**83.1**	**82.3**
snli	BERT	90.1	1.5	4.0	3.8
	RoBERTa	**91.2**	1.3	5.0	4.0
	ALBERT	87.6	0.2	1.5	1.8
	DistilBERT	88.2	0.8	2.5	2.9
	UniBERT	86.6	**21.5**	**18.7**	**17.7**
yelp	BERT	95.4	3.9	15.9	2.9
	RoBERTa	**96.6**	8.4	24.6	8.4
	ALBERT	94.7	1.3	5.9	1.4
	DistilBERT	95.9	4.9	17.4	2.6
	UniBERT	93.3	**75.3**	**79.6**	**75.9**

Table 4. Classification accuracies of our UniBERT vs. other baseline BERT models under attacks. We study the typographical (PWWS), typographical & synonym-based (Textbugger), and synonym-based (Textfooler) attacks on news categorization (ag_news), natural language inference (snli), and sentiment analysis (yelp) for a comprehensive evaluation on text classification robustness. In the table, we compare the post-attack accuracies of our UniBERT against four other BERT variants (namely, BERT, RoBERTa, ALBERT, and DistilBERT), showing that our UniBERT outperforms the baseline models by a large margin. Accuracy is reported as a percentage (out of 100) to measure the ratio of correct prediction. The best accuracies are bolded.

The unitary constraint also reduces model complexity. As a review of Sect. 2.2A earlier, the number of parameters (i.e., the degree of freedom) is about half of a fully connected layer when we count a real, adjustable scalar number as one parameter. Specifically, the number of Lie parameters is $n\ (n - 1)\ /\ 2$ for the orthogonal group of degree n, matching the number of Lie algebras that the group has (see Fig. 2). Consequently, we only need half as many real scalars to specify a neural layer under the unitary constraint. Because of the reduction in model complexity, UniBERT's original accuracy is closer to the reduced-size models (e.g., ALBERT and DistilBERT) as supposed to the full-size models (e.g., BERT). UniBERT's strong suit is at post-attack accuracies, at which it surpasses all models with large margins.

Neither the RoBERTa's enhanced pretraining nor ALBERT's or DistilBERT's model reduction results in significant improvements in the post-attack accuracy. With the data listed in Table 5, we confirm that adversarial attacks cannot be prevented with the baseline models alone. To counter adversarial attacks, further interventions are needed. As an example of such intervention, in UniBERT, we combine a wide decision margin with confined perturbations by using the multi-margin loss with unitary weights to deliver superior adversarial robustness.

B. Our UniBERT vs. Defense Models

We compare UniBERT's performance with the state-of-the-art defense techniques from the adversarial training and regularization categories outlined previously in Sect. 3.1C of the introduction. We call these neural networks designed with adversarial robustness in mind the "defense models." Namely, they are:

1. **A**dversarial and **M**ixup **D**ata **A**ugmentation (**AMDA-Tmix** & **AMDA-Smix**) [54]—In this technique, adversarial examples are generated by assuming a specific attack recipe. Furthermore, additional training data are created by linearly interpolating the neural representation and the label pairs. They designate their model with the "Tmix" and "Smix" postfixes to denote the location where the representation is taken from in BERT.

2. **M**ixup **R**egularized **A**dversarial **T**raining (**MRAT** & **MRAT+**) [55]—This technique uses all the steps detailed in AMDA above. In addition, it adds a regularization term in the loss function (i.e., a penalty for any data point that is too different from the original), ensuring that the augmented data will follow the original data distribution. They claim that this term will prevent the augmented dataset from degrading the pre-attack accuracy. Like AMDA, the key ingredient for improving robustness still comes from adversarial training with data augmentation. The MRAT+ variant adds data augmentation to the original data. For example, it may swap some words in the sentence with their synonyms to generate new samples.
3. **Info**rmation Bottleneck on **BERT** (**InfoBERT**) [56]—This technique reduces model complexity by using an information bottleneck. The bottleneck is implemented as two regularization terms in the loss function: one is to maximize the prediction accuracy while minimizing the mutual information between the input and the internal representation. The general concept of a loss function is explained in Sect. 3.2A; please refer to the InfoBERT paper for the complete mathematical description as there are many intricate details [56]. Another is to identify word embeddings that are less affected by the input perturbation and force the neural net to utilize these words more in its decision process. InfoBERT is computationally expensive due to the need to calculate mutual information. It also requires more hyperparameters, which makes model optimization cumbersome.

There are two ways to measure the post-attack classification accuracy: one is with targeted attacks; another is with static attacks as explained in Sect. 3.3E. Thus, we need to make sure that we use the same evaluation method to compare the adversarial robustness of different models. The authors of AMDA and MRAT have published the performance of their models using targeted attacks, and their data are reproduced in the table below for comparison. The creator of InfoBERT uses static attacks in their paper; thus, we rerun their simulations with the targeted attacks for a fair comparison. The adversarial performance of our UniBERT is always evaluated with the targeted attacks.

Task	Type	Model	Original Accuracy	Post-attack Accuracy		
				PWWS	Textbugger	Textfooler
ag_news	Adversarial Training [54]	AMDA-Tmix	**94.5**	69.7	N/R	56.3
		AMDA-Smix	94.3	70.0	N/R	51.3
	Regularization	InfoBERT	94.4	35.5	46.0	12.9
	Unitarity	**UniBERT**	92.3	**85.4**	**83.1**	**82.3**
snli	Adversarial Training [55]	MRAT	**89.5**	N/R	9.9	10.5
		MRAT+	88.7	N/R	12.2	12.4
	Regularization	InfoBERT	90.9	2.8	5.5	5.4
	Unitarity	**UniBERT**	86.6	**21.5**	**18.7**	**17.7**
yelp	Regularization	InfoBERT	**97.4**	3.9	21.0	2.1
	Unitarity	**UniBERT**	93.3	**75.3**	**79.6**	**75.9**

Table 5. Classification accuracies of our UniBERT vs. state-of-the-art defense models. AMDA-Tmix, AMDA-Smix, MRAT, and MRAT+ use adversarial training with mix-up data augmentation to deliver the previous state-of-the-art performance, and their data are reproduced here from [47] and [67], respectively. InfoBERT uses information bottleneck as a regularizer to slightly improve the post-attack accuracies. Out of all the defense techniques, UniBERT gives much higher post-attack accuracies over all attack recipes (PWWS, Textbugger, Textfooler). Accuracy is reported as a percentage (out of 100) to measure the ratio of correct prediction, and N/R indicates not reported in the original publication. The best accuracies are bolded.

Our UniBERT provides significant robustness enhancements compared to the other state-of-the-art defense methods, boosting the post-attack accuracy by at least 15.4% across all datasets and attack recipes. Under the targeted attacks, regularization-based models such as

InfoBERT only deliver marginal improvements over the classic BERT model. InfoBERT's robustness is slightly higher than RoBERTa's in natural language inference (snli), but it's worse in news categorization (ag_news) and sentiment analysis (yelp). On the other hand, AMDA, MRAT, and UniBERT deliver superior adversarial performance over InfoBERT. Furthermore, UniBERT is more robust than adversarial training, providing double-digit accuracy boosts (up to 31%) compared to AMDA in news categorization and single-digit improvements (up to 8.8%) compared to MRAT in natural language inference. This is because there is a wide variety of perturbations an attack recipe can produce for a given sentence; although adversarial training captures a decent portion of these variations with augmented data, it is impossible to be comprehensive and cover the entire search space. In practice, the designer of the neural network will not know the exact sentence that the user will enter. This is the reason that we separate the data into two mutually exclusive portions: the training set and the test set. The training set is used for training the neural net, and the post-attack accuracies are measured on the attacked test set for a realistic performance evaluation.

Our model has a slight degradation in the original accuracy but is still comparable to that of the adversarial training methods (AMDA-Smix and MRAT+). The key weakness of adversarial training is that the designers need to anticipate the types of attacks and create a representative adversarial training sample set. It is not always practical to obtain complete coverage of all attacks and may require significant training time. The advantage of UniBERT is that it does not assume which attack the hacker will use. Additionally, the robustness enhancement is generally consistent across various attack recipes (i.e., PWWS, Textbugger, and Textfooler) as shown in the post-attack accuracy columns in Table 5. Other defense models' performances vary greatly with the attack recipes as shown in the table.

C. Ablation Study

To show the contribution of the multi-margin loss and the unitary weights separately, we perform an ablation study for UniBERT on the yelp task. Table 6 reports the performance of the following four model trims:

1. **BERT** is the baseline with no modification.
2. **BERT_unitarity** is BERT with the unitary constraints alone.
3. **BERT_multi_margin** is BERT with the multi-margin loss instead of the cross-entropy loss without any unitary constraint.
4. **UniBERT** has both the unitary weights and the multi-margin loss replacement.

We observe that BERT_unitarity's post-attack accuracies are not significantly different from BERT (Table 6, first two rows), confirming our hypothesis that unitary weights must be used together with a large margin model. BERT_multi_margin delivers a significant improvement in the post-attack accuracy; however, it also reduces the original accuracy. By adding the unitary constraints on top of BERT_multi_margin, UniBERT raises both the pre-attack and post-attack accuracies considerably (Table 6, last row).

Task	Model	Original Accuracy	Post-attack Accuracy		
			PWWS	Textbugger	Textfooler
yelp	BERT	**95.4**	3.9	15.9	2.9
	BERT_unitarity	95.9	3.1	14.1	1.8
	BERT_multi_margin	82.9	63.0	63.2	54.8
	UniBERT	93.3	**75.3**	**79.6**	**75.9**

Table 6. Ablation study to understand the contribution of each component. BERT is the baseline model. BERT_unitarity adds unitary constraints to BERT without the multi-margin loss. BERT_multi_margin replaces the loss to multi-margin without placing any

unitary constraints. This data shows that we need to use the multi-margin loss together with unitarity to achieve the best post-attack accuracies in UniBERT. The multi-margin loss is used for increasing the decision margin while unitarity is for stabilizing perturbations. Accuracy is reported as a percentage (out of 100) to measure the ratio of correct prediction. The best accuracies are bolded.

With this ablation study, we conclude that it is insufficient to have unitarity alone; the multi-margin loss and unitarity need to be used together to achieve the best result. The reason is the following: In the case of adversarial attacks with infinite query budgets, the attackers are free to try as many permutations as possible until they exhausted all combinations. A query budget is defined as the number of attempts the attacker is allowed to make before giving up. Because the post-attack accuracies reported in this dissertation are measured with attacks with infinite query budgets, the attacker has a large search space to find one adversarial example that changes the prediction outcome, and it only needs one example for the attack to count as successful. Besides, the embeddings for synonyms are more likely to be similar but their proximity in the vector space is not guaranteed since the model (such as UniBERT) learns the embeddings automatically through backpropagation during pretraining. Hence, because both synonym swaps and typos have a small chance of producing large perturbations in the sentence embedding, a large perturbation will occur with a high probability given enough samples.

In short, attacks with unlimited budgets can create large perturbations that cross the decision boundary; hence, the networks must use the multi-margin loss to create sufficient decision margins to absorb such perturbations (as shown in Table 6, BERT_multi_margin). Albeit unitarity does not suppress the perturbations, it at least eliminates the possibility of accidentally amplifying them. Its norm preserving property is proven earlier in Eqn. (15). With an unlimited query budget, large perturbations are bound to happen; therefore, unitarity alone will not prevent the logits to cross the decision boundary if the decision margin were too small. Unitarity is only helpful when used in conjunction with the large decision margin created by the multi-margin loss; thus, we set ε to 100 in our experiments. Contrarily, non-unitary networks may amplify perturbations, resulting in misclassification; thus, they have lower post-attack accuracies (see Table 6, BERT_multi_margin).

D. Effect of the Multi-margin Loss

To confirm that the multi-margin loss widens the decision margin, we study the logits in our UniBERT for 950 correctly labeled data samples with the yelp binary classification task, compared with the ones in BERT. Logits are the activations of the last neural layer in the neural net as defined earlier in Sect. 3.1A. More precisely, for each data sample, we measure the logits' shortest distance to the decision boundary in the vector space. Samples with the wrong prediction outcome are ignored because the attack algorithms skip any misclassified sentences, so they will never be attacked.

Model	Unitary	Loss	Average Decision Margin (a.u.)
BERT	No	Cross-entropy	6.1 ± 1.1
UniBERT	Yes	Multi-margin	72.4 ± 2.1

Table 7. Average distance to the decision boundary for the logits. We define decision margin as the average distance between the logits to the decision boundary for correct samples. The number after the ± sign is the standard deviation. The UniBERT increases the decision margin by widening the gap between the neural representations of different classes while keeping intra-class spread (i.e., the standard deviation) small, encouraging distinct representations for each class.

We report the average distance measured in all 950 samples in Table 7, and we name the average distance to the decision boundary the decision margin. In comparing the decision margins between different models, it is important to evaluate not only the mean but also the standard deviation of the decision margin to consider the simple scaling of the logits. UniBERT's logits are much further away from the decision boundary compared to BERT's (71.5 vs. 5.8) as shown in Table 7. This 12X increase is not the result of a simple scaling of the logits because the standard deviation stays roughly the same (2.4 vs. 1.4); thus, they are statistically well separated in UniBERT compared to BERT. With the yelp dataset, our model is asked to categorize the input text into a positive review or a negative review; hence, the dataset has two classes. Another way to interpret this data is that UniBERT increases the inter-class distance between the neural distributions for different classes while deterring the intra-class spread within the same class.

E. Propagation of Perturbation

The most important benefit of the unitary weights is that they regulate the magnitude of a perturbation throughout the network as proven mathematically in Eqn. (15); nevertheless, a completely unitary neural network is impossible because non-square weights and nonlinearities are needed for creating the required model complexity to solve complex problems as discussed in Sect. 3.3A previously. In our last experiment as follows, we demonstrate that, even with imperfect unitarity, our UniBERT can still stabilize the perturbations, preventing them from being scaled arbitrarily by the synaptic weights.

To quantify the effect of a perturbation, we use a mathematical concept called cosine similarity. *Cosine similarity* measures the alignment of two vectors, which is defined as:

$$cosine\ similarity \triangleq \frac{\bar{a} \bullet \bar{b}}{\|\bar{a}\|_2 \|\bar{b}\|_2} = \cos\theta, \tag{20}$$

where $\bar{a}$ and $\bar{b}$ are two vectors in the same vector space, • is the vector dot product, and θ is the angle between $\bar{a}$ and $\bar{b}$. Cosine similarity ranges from -1 to 1, inclusive. When the two vectors overlap completely, the cosine similarity is the highest, signaling that the two vectors are identical. If they point in opposite directions, the cosine similarity is -1. In the context of word embeddings, researchers often use it to measure how close two words are in meaning by using the cosine similarity between the two corresponding word vectors (i.e., the vectors that encode the words).

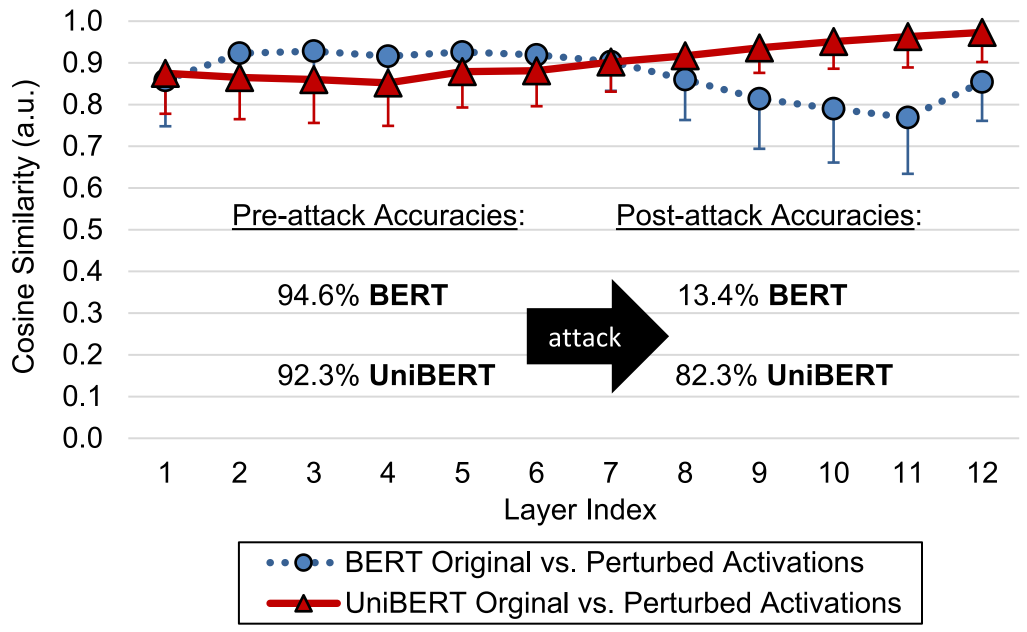

Fig. 11. Cosine similarity between the original activations and the perturbed activations under attack. We measure the cosine similarity between the activations of the original sentence and the activations of the perturbed sentence in UniBERT at the output of each attention layer (i.e., $\bar{x}_1 \ldots \bar{x}_{12}$ in Fig. 9) indexed from 1 to 12 across the neural network (dotted line). The same is done for BERT (dashed line). As defined in

Eqn. (20), cosine similarity ($\in\mathbb{R}\cap[-1,1]$) is a distance measure for quantifying how well two vectors align with each other in a vector space with one indicating that the two vectors are identical. For UniBERT, the representations become more alike in deeper layers; on the contrary, the similarity fluctuates in BERT. A higher cosine similarity score means that the network is capable of restraining perturbations closer to the original representation; consequently, higher post-attack accuracies (see the text added in the figure). We perform statistical analysis on 1000 randomly chosen sentences; the error bars denote standard deviation.

Likewise, we use cosine similarity to quantify how much the activations deviate from their original values when the network is under attack. If the network is less susceptible to the input perturbations injected by the attacker, the attacked activations will be similar to its original, yielding a high cosine similarity. To visualize the neural network model's susceptibility to injected perturbation, we measure the cosine similarity between the perturbed activations and the original activations across the neural net and plot the results in Fig. 11 above. In particular, we randomly select 1000 sentences from the ag_news dataset, produce perturbed sentences using the Textfooler attack recipe, and record the activations after each of the 12 unit blocks in UniBERT (see Fig. 9 for the network architecture). We perform the same steps for BERT and compare the two results.

As shown by the dotted line in Fig. 11, BERT's cosine similarity fluctuates wildly across the different layers of the networks. In particular, it dips to the lowest value at layer 11 with a mean (μ) of 0.77 and a standard deviation (σ) of 0.14. In a normal distribution, 16% of the samples reside below one standard deviation from the mean (i.e., $\mu - \sigma$); likewise, 34% of the samples fall between the mean (μ) and the first standard deviation ($\mu - \sigma$). The activations at layer 11 deviate from their original values significantly when BERT is under attack, and the cosine similarity quantifies this deviation: 16% of the 1000 samples results in a cosine similarity score of less than 0.63 (i.e., $\mu - \sigma$, or 0.77 – 0.14), and 34% of them are between 0.63 and 0.77. In short, when BERT is under attack, a significant portion of the sentences have activations that are a great deal different from their original values. On the other hand, our UniBERT fixes this problem by improving the similarity between the original and attacked activations (Fig. 11).

This result is interesting because only 62.3% of the weights in UniBERT are unitary and there are many nonlinearities in the network (see Sect. 3.3A for architectural details). We argue that the success of UniBERT in stabilizing the perturbation comes from the unitarity added in the attention layers, which are repeated 12 times in the network. The attention mechanism is achieved primarily with the Key, Query, Value, and Dense modules in Fig. 9**;** the weights in these modules are all unitary in UniBERT as shown in Table 1. Although the decision processes in a neural net are generally non-interpretable, with the data in Fig. 11, we conclude that the novel unitary attention mechanism in our UniBERT encourages the network to make similar decisions even under adversarial attacks as shown with the rising cosine similarity of the activations shown in the figure for UniBERT (Fig. 11, solid line).

It is difficult to exactly predict the post-attack accuracies from the cosine similarity between activations, but the two quantities are positively correlated. A higher cosine similarity means that the activations under attack are similar to the original ones; therefore, the network is more likely to achieve the pre-attack accuracies as if there is no attack because the logits are less likely to cross the decision boundary. For reference, we insert the pre-attack and post-attack accuracies in Fig. 11. UniBERT has higher cosine similarity compared to BERT in layers that are immediately proceeding to the final classifier (i.e., layers 8-12 in Fig. 11). The resilience to perturbation is especially important later in the network where decisions are made by comparing the final activations (logits) to the decision boundary, and we observe that UniBERT can deliver a superior post-attack accuracy (82.3%) compared to BERT's post-attack accuracy (13.4%) by keeping its activations closer to their original values before the attacks.

3.5. Conclusion

In this chapter, we present an enhanced neural architecture named UniBERT for robust natural language processing. Our UniBERT defends against adversarial attacks, in which the adversaries attempt to slightly modify the input sentences to disrupt the prediction results in deep neural nets. It is a variant of BERT with improvements including replacing the cross-entropy loss with the multi-margin loss during finetuning to increase the decision margin and forcing unitary constraints on the attention layer to restrain the perturbations injected during an attack. With UniBERT, unitary weights with multi-margin loss are shown to be an effective defense technique against both typographical and synonym-based adversarial attacks, boosting the post-attack classification accuracy by more than 15.4% compared to the state-of-the-art defense mechanisms across all datasets and attack recipes (Sect. 3.4B).

In addition, we study the contributions from multi-margin loss and unitarity in UniBERT individually, discovering that unitarity has to be used together with the multi-margin loss to be effective (Sect. 3.4C). Because we allow an unlimited number of attempts until the attacker has exhausted the search space, some perturbations will be large in our experiments. Thus, the multi-margin loss needs to first create a large decision margin (Sect. 3.4D) capable of deterring most logits from crossing the decision boundary before the effect of unitarity in stopping accidental amplification of perturbations can be seen. Also, unitarity stabilizes perturbations by keeping the activations close to their original values across the network (Sect. 3.3E). Orthogonal to the state-of-the-art defense methods (e.g., adversarial training or regularization), our UniBERT is straightforward to implement and works well for a wide range of applications including categorization, natural language inferencing, and sentiment analysis. Our contribution will empower practitioners to build efficient, robust, and safe NLP pipelines for critical applications.

Chapter 4 — Quantum Unitary Neural Nets

A primer for the next technology curve.

4.1. Introduction

A. The Quantum Transition

Semiconductor innovations enabled our computational capability to scale exponentially for many decades [8]. Shrinking the physical dimensions of transistors and interconnects is vital for the technology industry because we can have cheaper, faster, and more powerful chips with smaller feature sizes. Scaling in the nanometer regime is an intricate dance choreographing lithography, leakage, reliability, and yield. Despite our past success, the introduction of newer technology nodes has slowed down in recent years due to the difficulties in manufacturing nanometer-sized transistors uniformly at scale [78]–[80]. Thus, our industry needs a disruptive hardware technology to establish further scaling beyond the complementary metal-oxide-semiconductor (CMOS) devices.

Quantum computing is a promising candidate because the amount of information that a quantum computer can compute at a given instance is exponential to the number of quantum bits (qubits) regardless of the devices' physical sizes. There are many variants of quantum computers, and the one we will study here is the quantum circuit model [81]. It is a non-dissipative system that uses quantum gates to process a massive amount of information in parallel. A quantum computer is a linear system whose dynamics are governed by the time-dependent Schrödinger equation:

$$i\hbar \frac{d}{dt} |\Psi(t)\rangle = \widehat{\boldsymbol{H}} |\Psi(t)\rangle, \tag{21}$$

where $\widehat{\boldsymbol{H}}$ is the Hamiltonian operator. Here we assume $\widehat{\boldsymbol{H}}$ to be time-independent in our system. The solution of the Schrödinger equation describes how the system propagates in time, given the initial state $|\Psi(t)\rangle$ at $t = 0$:

$$|\Psi(t)\rangle = e^{-i\widehat{\boldsymbol{H}}t/\hbar}|\Psi(0)\rangle = \widehat{\boldsymbol{U}}|\Psi(0)\rangle. \quad (22)$$

We name $\widehat{\boldsymbol{U}}$ the time-evolution operator, which is a unitary matrix.

A quantum neural network (QNN) is a neural net implemented in a quantum computer to leverage the benefit of the exponential number of states for a potential computational speedup via parallel computation. In a QNN, we encode the neurons' activations with the quantum states and implement synaptic weights with quantum gates. These quantum gates are unitary transforms that define the time-evolution operator $\widehat{\boldsymbol{U}}$ in Eqn. (22); we design them intentionally to bring the system from an initial state to the desired final state. The quantum computer we study herein is a linear system because it follows the Schrödinger equation, which supports the superposition principle of wavefunctions. We will utilize this linear, non-dissipative, and unitary quantum computer as a subcomponent of a hybrid QNN. The state-of-the-art QNN is still very limited in its functionality because nonlinearity needs to be implemented using ancillary qubits or measurements. Applications for QNN today are limited to image recognition of greyscale images with small image resolutions.

B. <u>The Problem of Coherence Time</u>

Unlike classical transistors, qubits interact with one another. This interaction is called entanglement, and qubits need to stay coherent to achieve entanglement [82], [83]. There are two ways to quantify coherence [84]: the first one is called single qubit relaxation time (T_1). It measures how long a qubit stays in the excited state before relaxes down to the ground state. The second is a multi-qubit phase coherence time (T_2), which measures how long can two qubits stay in sync with each other. These metrics are critical because quantum information is often encoded in the entangled states, and losing coherency means that information will be lost.

Today, despite a tremendous amount of effort in developing quantum devices, we are still highly constrained by the coherence time. For illustration, IBM's flagship quantum computer has a minimum T_1 of 38 μs and a minimum T_2 of 39 μs [85]. With the assumption that a single gate time is roughly 0.5 μs, the quantum computer will become decoherent after 76 gate operations. Consequently, coherence time is a sacred resource, and we must use it efficiently. A more complex neural network will require more quantum gates, and therefore, it needs a quantum computer with a longer coherence time to run. The designer will be interested to build a network that has the minimum complexity required to solve a particular problem. Nevertheless, there is currently no framework to estimate the required model complexity.

C. Our Proposed Solution

There is an intuitive relationship between the complexity of the problem and the required complexity of the machine learning model. If the problem is more complicated, it requires a fitting function to have higher degrees of freedom. In other words, complex problems need larger models with more tunable parameters. In this work, we discover a way to relate the two—By decomposing the dataset using the principal component analysis (PCA), we can calculate the intrinsic dimensionality of the input signals and use it to guide us in estimating the number of quantum layers needed to solve a classification problem.

In machine learning, an input datum can be presented using a vector of *n* dimensions, and we can plot the entire dataset as points in an *n*-dimensional vector space. PCA identifies the important directions in this vector space along which the variance is maximized, providing a way for us to estimate the intrinsic dimensionality of the dataset. The exact mathematical formulation of PCA will be explained in Sect. 4.2C. We show in theory and by simulation that the image reconstruction accuracy vs. the number of eigenvectors used in PCA correlates with the classification accuracy (ξ) of the quantum system with corresponding circuit depth (k). The classification accuracy (ξ) is defined as the percentage of samples correctly recognized by our machine learning system, which consists of both quantum and classical components with details to be followed. In addition to predicting the minimum required number of quantum gates, our framework offers a quick way to approximate the tradeoff between classification accuracy (ξ) and depth (k), which is invaluable to machine learning practitioners in architecting their QNN without requiring them to perform the lengthy hyperparameter tuning procedures. Our framework is a practical rule of thumb for designing QNN in the noisy intermediate-scale quantum (NISQ) era [86].

D. Literature Review

There is a quantum renaissance in recent years. In 2019, Havlicek et al. proposed using quantum computers as high-dimensional kernel functions for classical support vector machines [87]; this work uses a 5-qubit superconducting quantum computer consisting of Josephson junctions to classify a synthetically generated small dataset. Even though this work mainly focuses on training the quantum features, it establishes an excellent network architecture for multiclass classification in quantum computing. In 2021, Jiang, Xiong, and Shi devised a quantum neural network for image classification [88]. Their proposed architecture uses binary synaptic weights to perform binary classification on 4 x 4 pixels greyscale images, and they show that quantum supremacy is possible. The same authors also wrote a perspective paper on the idea of hybrid quantum machine learning that introduces classical computers as data coprocessors [89].

Moreover, the Vapnik-Chervonenkis dimension (or the "VC dimension" as commonly referred to in the computer science literate) quantifies the intellectual capacity of a machine learning model by the maximum number of data points it can perfectly separate in a binary classification task. For instance, if we say that a neural net has a VC dimension of 5, it means that its decision boundaries are complicated enough to perfectly separate 5 data points arranged in any configuration (i.e., these 5 points can be placed in any locations in the vector space with arbitrary binary class labels). Chen et al. published an academic study using fictional datasets for quantifying the model capacity of quantum circuits [90]. They offered theoretical derivations utilizing the Vapnik-Chervonenkis dimension to estimate the requirement on the circuit depth. Despite having profound academic merits, their theory only works in the case where the number of qubits is larger or equal to the dimensionality of the feature space; hence, it is unlikely to be practical in realistic settings.

4.2. Theory

A. Hybrid Quantum Neural Net

A hybrid QNN [89] leverages the power of both classical and quantum computing: The classical computer handles nonlinear feature extraction, and the quantum computer carries out linear fitting. Nonlinearity is essential in neural nets because it enables elaborate decision boundaries for complex analytics. On the contrary, if all layers of a neural net are linear, the entire network can be collapsed into one linear transformation similar to how matrix multiplications can be combined into one. In this case, model capacity is greatly diminished. At present, quantum computers are linear. Although it is possible to implement a square nonlinearity during measurement when the wavefunctions collapse, this prohibits further quantum processing and thus foregoes the benefit of the exponential number of states. The square nonlinearity arises from the fact that the probability for a qubit to be observed in a particular state is the square of its amplitude. We will discuss the possibility of a nonlinear quantum computer in Chapter 5 that enables end-to-end quantum computations for neural networks. But for now, we stick to the confines of the current quantum technology. Our hybrid QNN uses a classical computer to extract the nonlinear features and a quantum computer to linearly rotate the decision boundary to fit the data. Fig. 12 illustrates a hybrid QNN with three qubits.

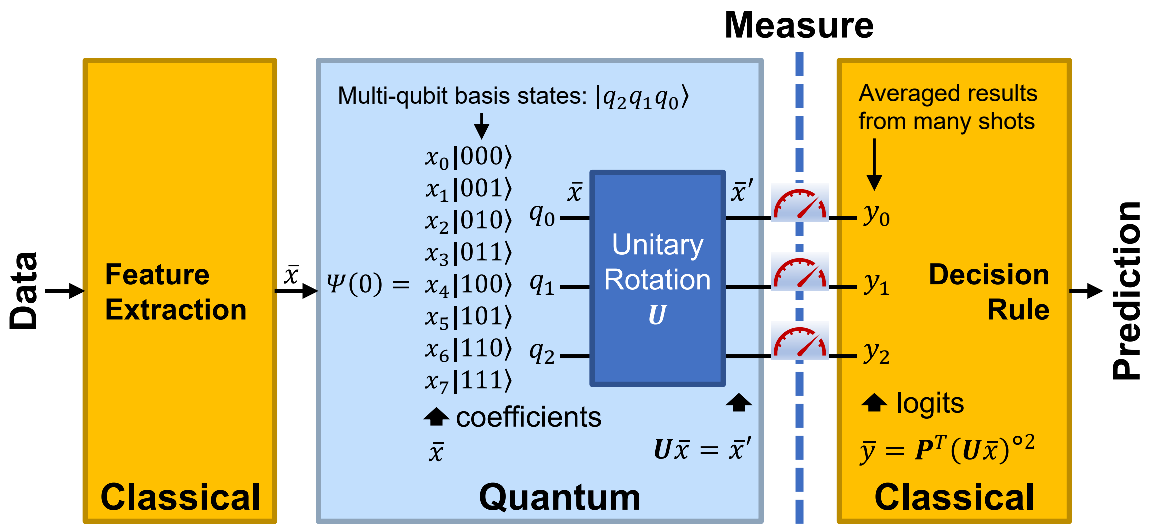
Measure
Data
Feature Extraction
Classical
$\bar{x}$
Multi-qubit basis states: $|q_2q_1q_0\rangle$
$\Psi(0) = x_0|000\rangle\ x_1|001\rangle\ x_2|010\rangle\ x_3|011\rangle\ x_4|100\rangle\ x_5|101\rangle\ x_6|110\rangle\ x_7|111\rangle$
q_0
q_1
q_2
$\bar{x}$
Unitary Rotation U
$\bar{x}'$
coefficients
$\bar{x}$
$U\bar{x} = \bar{x}'$
Quantum
Averaged results from many shots
y_0
y_1
y_2
Decision Rule
logits
$\bar{y} = P^T(U\bar{x})^{\circ 2}$
Classical
Prediction

Fig. 12. Our proposed hybrid quantum neural network. Data flow through multiple subsystems to arrive at a prediction. The feature extraction step on the left can be implemented as a classical neural network that incorporates nonlinear functions. The quantum step in the middle depicts an example quantum computer with three qubits, rotating the data with $\boldsymbol{U}$ to fit the decision boundaries predetermined by the measurement setup and the decision rule. In each experiment, the act of measurement collapses the high-dimensional quantum data, $\boldsymbol{U}\bar{x}$, to a classical vector, $\bar{y}_{single\_shot}$ (not shown). Averaging $\bar{y}_{single\_shot}$ over many identical experiements (i.e., shots) produces the logits, $\bar{y}$, whose values can be predicted by Eqn. (23) and as shown on the bottom-right of this figure.

As shown in Fig. 12, the proposed hybrid QNN has three components: a feature extractor in a classical computer, a quantum component that implements a unitary rotation ($\boldsymbol{U}$) in the high-dimensional space, and a classical post-processing portion to convert the quantum states into a prediction. The purpose of feature extraction is to sift through useful information in the data with a nonlinear function. As mentioned, nonlinearity is essential in creating elaborate decision boundaries [11] because if there is no nonlinearity between the layers, the entire network can be condensed into a single linear transformation like combining many matrix multiplications into one. The nonlinear feature extractor complements the subsequent linear quantum computer, and extracted features are flattened into a vector and encoded as the quantum state amplitudes in the quantum computer. The goal of the quantum computer is to rotate the high-dimensional vector ($\bar{x}$) with $\boldsymbol{U}$ to fit the decision boundaries, where $\bar{x}$ is encoded in the coefficients of the multi-qubit computational basis states (Fig. 12, middle). After the rotation, the quantum states are measured using the Pauli-Z operator, which has eigenvalues of ±1 [91]. That is, the final output of a single quantum measurement yields a vector of ±1s, and the size of this classical vector is the same as the number of qubits, n_c. We can compute the logits ($\bar{y}$) shown on the right of Fig. 12 by averaging the measurement results from many identical quantum experiments, which are called shots in quantum computing. As a review of our discussion in Sect. 3.1A, the term *logits* refers to the activation values of the last layer in a neural network (i.e., they are a neural net's output values arranged in a vector form). Since the quantum computer in Fig. 12 implements a quantum neural network, we refer to its average output ($\bar{y}$) as logits to connect with deep learning literature.

The logits ($\bar{y}$) are related to the amplitudes of the initial quantum states ($\bar{x}$). $\bar{y}$ is stored in a classical computer indicating the averaged experiment outcome while $\bar{x}$ is the coefficients of the wavefunctions in a quantum computer. The transfer function for our quantum system including the measurement process (Fig. 12) can be described mathematically as:

$$\bar{y} = \boldsymbol{P}^T(\boldsymbol{U}\bar{x})^{\circ 2}, \tag{23}$$

where $\circ 2$ is the Hadamard power[13] that squares each element individually, $\bar{y} \in \mathbb{R}^{n_c}$ is the average measurement results, $\boldsymbol{P}^T \in \mathbb{R}^{n_c \times n_s}$ is the classical projection matrix for the measurement (with detailed descriptions to be followed), $\boldsymbol{U} \in \mathbb{R}^{n_s \times n_s}$ is the unitary rotation, $\bar{x} \in \mathbb{R}^{n_s}$ is a vector representing amplitude coefficients for the initial state, n_s is the number of quantum states, and n_c is the number of qubits (representing the number of classes). The number of quantum states is exponential to the number of qubits:

$$n_s = 2^{n_c}. \tag{24}$$

Although $\boldsymbol{P}^T$ is not a quantum mechanical operator, $\boldsymbol{P}^T$ is constructed from the quantum states while considering the eigenvalues of the Pauli-Z operator. We call $\boldsymbol{P}^T$ the projection matrix because it projects the probability of being in each quantum state, an n_s-dimensional vector, down to an exponentially smaller n_c-dimensional qubit subspace, succinctly capturing the effect of averaging a large number of measurement results. In this example with three qubits, we write

$$\boldsymbol{P}^T = \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}. \tag{25}$$

We construct the columns of $\boldsymbol{P}^T$ as follows: First, we list out all the multi-qubit basis states (i.e., |000>, |001>,…, |111>). The wavefunction collapses into one of the basis states (e.g., |000>) after we measure the quantum system. We read out the corresponding eigenvalues of the measurement operators (e.g. [-1, -1, -1] for Pauli-Z). Each row signifies the direction of

[13] The Hadamard power performs element-wise exponentiation on each matrix or vector element. For instance, $\boldsymbol{B} = \boldsymbol{A}^{\circ n}$ means that $b_{ij} = a_{ij}^n \ \forall \ (i,j)$ where b_{ij} is an element of **B**; a_{ij}, an element of **A**.

projection in the n_s-dimensional vector space. Depending on the measurement operator (e.g., Pauli-Z or others) used for reading out the quantum state, $\boldsymbol{P}^T$ can take on different values; however, it will always be n_c x n_s. We will use the logits ($\bar{y}$) to predict the class label in a classification task as we will explain next.

B. <u>Decision Boundaries</u>

We use the logits ($\bar{y}$) to decide which class $\bar{x}$ belongs to. In a classification task, each datum belongs to one of the n_c classes, and the goal of our machine learning system is to predict the correct class label for a given input. We deliberately design our hybrid QNN to have the same number of qubits as the number of classes so that measuring the wavefunction will result in a vector of size n_c. Finally, we simply use the argmax function to pick the index i that has the highest logit (y_i) and predict that the data belongs to class i:

$$predicted\ class = \underset{i}{\operatorname{argmax}}\, y_i, \tag{26}$$

where y_i is a vector element of $\bar{y}$ and i takes on all integer values from 0 to $n_c - 1$, inclusive. We refer to Eqn. (26) as the decision rule, which prescribes the way we predict the class label given the logits ($\bar{y}$). It creates multiple decision regions that separate the feature space. A decision region is an area in the activations vector space, where all the data belong to the same classification. We can derive the equations for the boundaries of the decision region (i.e., the decision boundaries), where there is an equal probability of predicting class i versus class j. When we use the argmax decision rule, the number of boundaries (n_b) in the n_c-dimensional logits space is governed by the n-choose-2 combinatorial formula, $C_{n_c}^2$, because it is equivalent to comparing any two of the n_c logits:

$$n_b = C_{n_c}^2 = \frac{n_c(n_c-1)}{2}. \quad (27)$$

In our three-qubit example, three boundaries separate the three decision regions. The first boundary separates class 2 and class 1. For clarity, we rewrite Eqn. (23) in the scalar form as follows:

$$y_0 = -{x'_0}^2 - {x'_1}^2 - {x'_2}^2 - {x'_3}^2 + {x'_4}^2 + {x'_5}^2 + {x'_6}^2 + {x'_7}^2 \quad (28)$$

$$y_1 = -{x'_0}^2 - {x'_1}^2 + {x'_2}^2 + {x'_3}^2 - {x'_4}^2 - {x'_5}^2 + {x'_6}^2 + {x'_7}^2 \quad (29)$$

$$y_2 = -{x'_0}^2 + {x'_1}^2 - {x'_2}^2 + {x'_3}^2 - {x'_4}^2 + {x'_5}^2 - {x'_6}^2 + {x'_7}^2, \quad (30)$$

where x'_i is the amplitude of the ith quantum state after the unitary transformation (i.e., $\bar{x}' = \boldsymbol{U}\bar{x}$ and x'_i is the ith element of $\bar{x}'$). By equating *y0* in Eqn. (28) to *y1* in Eqn. (29), we derive the formula for the decision boundary between class 0 and class 1, where on this boundary, there is an equal probability of being either of the two classes. We obtain,

$${x'_4}^2 + {x'_5}^2 = {x'_2}^2 + {x'_3}^2. \quad (31)$$

Similarly, for the boundary between class 1 and class 2, we equate *y1* to *y2* and show that:

$${x'_2}^2 + {x'_6}^2 = {x'_1}^2 + {x'_5}^2. \quad (32)$$

Lastly, the boundary between class 2 and class 0 is derived by equating *y2* to *y0*:

$${x'_4}^2 + {x'_6}^2 = {x'_1}^2 + {x'_3}^2. \quad (33)$$

The decision boundaries in the three-qubit QNN have a cone-shape geometry in the high dimensional space as demonstrated by Eqns. (31)-(33).

The role of the unitary rotation (***U***) is to rotate the data to fit the boundaries since the boundaries are fixed with respect to the standard coordinate in $\bar{x}'$. To visualize this rotation, we can imagine, for instance, that the input data are Gaussian distributions spread out in a high dimensional space, in which each object class has a different mean and variance. As we discussed in Sect. 2.2B, ***U*** can be understood as rotating the data points in a fixed coordinate system or rotating the coordinate system while keeping the data in place. Hence, ***U*** rotates and moves the data into the two decision boundaries, or equivalently, it rotates the two decision boundaries to fit the data into the region. During the training of our hybrid QNN, the learning algorithm is responsible for finding the most optimal ***U***, resulting in a set of decision boundaries that best separate different classes of data to maximize the prediction accuracy. In the following, we will investigate how to implement ***U*** most efficiently in quantum computing.

C. Minimal Degrees of Freedom

This subsection aims to establish the minimal degrees of freedom that a QNN needs for solving a classification problem. The computational complexity of a quantum algorithm is proportional to the number of quantum gates that it requires, and each elementary quantum gate provides a rotational degree of freedom. We define the elementary quantum gate as any single-qubit gate with an adjustable angle parameter. For example, the R_y gate is the elementary gate that we use in this study for quantum complexity:

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{34}$$

where θ is the azimuthal angle in the Bloch sphere. The half-angle ($\frac{\theta}{2}$) in Eqn. (34) has profound meaning in physics, and we will discuss it here briefly. The Bloch sphere is a 3D representation of the state of a single qubit with the north pole representing the spin-up state of an electron and the south pole representing the spin-down state. Succinctly put, electron spins have SU(2) symmetry with a periodicity of 4π but the Bloch sphere has SO(3) symmetry with a periodicity of 2π; to match the two, we need to make the azimuthal angle (θ) in the Bloch sphere rotate at 50% speed such that a rotation of 2π in the longitude of the Bloch sphere represents a 4π rotation in the spin space [92]. The number of thetas (θ) we need to specify $\boldsymbol{U}$ determines the degrees of freedom in our machine learning model. The QNN is free to adjust these angles to maximize the prediction accuracy during training, and these angles will be constant during inference. Since coherence time is a scarce resource today, it is ideal to limit the number of quantum gates in our model to the minimum required by the classification problem.

To estimate the complexity of the problem, we use the principal component analysis (PCA) as described before to identify the intrinsic dimensionality of the dataset [93]. PCA decomposes the covariance matrix of a dataset using the singular value decomposition (SVD) [12]. In SVD, we decompose a rectangular matrix ($\boldsymbol{X}$) into three components ($\boldsymbol{V}$, $\boldsymbol{S}$, and $\boldsymbol{Q}$) as follows:

$$\boldsymbol{X} = \boldsymbol{V}\boldsymbol{S}^T\boldsymbol{Q}^T, \tag{35}$$

where $\boldsymbol{X} \in \mathbb{R}^{n_s \times m}$ is a wide rectangular matrix where each column is a data sample (i.e., the $\bar{x}$ from Eqn. (23)), $\boldsymbol{V} \in \mathbb{R}^{n_s \times n_s}$ is a unitary matrix representing the eigenvectors, and $\boldsymbol{S}^T \in \mathbb{R}^{n_s \times m}$ is a diagonal matrix representing the singular values, $\boldsymbol{Q}^T \in \mathbb{R}^{m \times m}$ is another unitary matrix, and m is the number of samples in the dataset. PCA relates to SVD via the eigenvectors ($\boldsymbol{V}$) and singular values ($\boldsymbol{S}$):

$$\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{V}\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T, \tag{36}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with elements being the eigenvalues of the covariance matrix, $\boldsymbol{XX}^T$. Like image compression based on Fourier analysis, we can compress our dataset by keeping the largest k eigenvalues and their corresponding eigenvectors, and the quality of this data compression technique based on PCA will depend on the distribution of the eigenvalues. We can use the reconstruction error as a quality metric described next.

Next, we will show that for our proposed QNN, $\boldsymbol{U}$ only requires k x n_c parameters for manipulating the top k PCA eigenvectors in a classification problem with n_c classes. To understand this statement, we break the derivation into two parts—The first parameter reduction comes from the fact that the dataset is compressible, and the second parameter reduction is due to the wavefunction collapse during measurement. Firstly, the transfer function of our QNN can be written in the following matrix form concerning the entire dataset by substituting Eqn. (35) into $\boldsymbol{Y}$ so that:

$$\boldsymbol{Y} = \boldsymbol{P}^T(\boldsymbol{UX})^{\circ 2} = \boldsymbol{P}^T(\boldsymbol{UVS}^T\boldsymbol{Q}^T)^{\circ 2}, \tag{37}$$

where $\boldsymbol{Y} \in \mathbb{R}^{n_c \times m}$ represents the average measurement results (i.e., each column in $\boldsymbol{Y}$ is the logits $\bar{y}$ from Eqn. (23)). Now we define $\boldsymbol{B}$ as:

$$\boldsymbol{B} = \boldsymbol{UV}, \tag{38}$$

where $\boldsymbol{B}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ are unitary matrices of size n_s x n_s. Only the top k rows in $\boldsymbol{U}$ have significant effects on the output because they correspond to the k first columns in $\boldsymbol{B}$, which will eventually multiply the top k singular values. Additionally, the rows (and columns) of a unitary matrix such as $\boldsymbol{U}$ are orthogonal. The top k rows of $\boldsymbol{U}$ have k x n_s matrix elements, and after transposition, they belong to the Stiefel manifold, which we discussed previously in Sect. 2.2B. Geometrically, a Stiefel manifold is a subspace of the manifold formed by the orthogonal group. An orthogonal matrix specifies the complete n_s basis vectors in an n_s-dimensional space, and taking k of these vectors will form the Stiefel manifold. Because we use R_y gate exclusively in our network, $\boldsymbol{U}$ is a real matrix in our case, and our previous analysis on the number of tunable scalar parameters still holds: As shown in Fig. 3 previously for the construction of a Stiefel matrix, a Stiefel manifold of size n_s x k can be represented by k x n_s – k x (k + 1) / 2 tunable parameters (i.e., angles) [94]. When n_s >> k, the first term dominates, and we simplify the expression to k x n_s parameters. So far, we have shown that $\boldsymbol{U}$ requires k x n_s parameters to program due to the compressibility of the dataset.

Secondly, we will explain the effect of quantum measurement to further zero in on the required number of parameters. The act of measurement collapses the quantum states with the projection matrix, $\boldsymbol{P}^T$ in Eqn. (25), which performs a linear transformation and projects the probability vectors from an n_s-dimensional space to an n_c-dimensional subspace:

$$\boldsymbol{P}^T : \mathbb{R}^{n_s} \rightarrow \mathbb{R}^{n_c}. \tag{39}$$

The target subspace is spanned by the columns of $\boldsymbol{P}$ as its basis, and any component perpendicular to the target subspace will be annihilated during measurement. In other words, while $\boldsymbol{U}$ can rotate the signal in n_s dimensions, only the rotations in the n_c-dimensional target subspace will affect the final output. Consequently, we can further reduce the parameter requirements from k x n_s to k x n_c with k being the minimum number of PCA components needed to represent a dataset, which we will quantify later experimentally. k x n_c is the minimal degree of freedom (i.e., tunable parameters) that our quantum computer must provide to solve a classification task with n_c classes. We will confirm our theory with quantum simulation in Sect. 4.4 below.

4.3. Method

A. Network Architecture

To verify our theory regarding the number of parameters required to specify the unitary rotation (***U***) for the proposed hybrid QNN, we expand our framework in Fig. 12 to ten qubits and apply it to solve an image classification problem. We illustrate the complete network architecture in Fig. 13 below. The classical feature extraction step pre-processes the input greyscale image (32 x 32 pixels) with a single convolutional layer, which has a 3 x 3 filter, 3 biases, and a padding of 1 and is labeled as Conv3x3 in Fig. 13 at the top. The resulting signal passes through the rectified linear unit (ReLU) to achieve a nonlinear transformation [15]. We flatten the signal to a vector of size 1024 (2^{10}) and initialize a 10-qubit quantum computer accordingly such that the quantum state amplitudes are equal to the vector. This process is labeled as the amplitude encoder in the figure.

The most important part of our proposed hybrid QNN is ***U*** in Eqn. (23) described in Theory Section 4.2 and depicted in Fig. 13. ***U*** has l layers, and each layer consists of 10 R_y rotation gates and 10 CNOT gates. CNOT is the controlled-NOT gate [91], defined as follows for a two-qubit system with the convention that the state vector is arranged from the least significant bit (|00>) to the most significant bit (|11>) from top to bottom:

$$\boldsymbol{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{40}$$

Also, we construct the matrix representing CNOT in Eqn. (40) with the assumption that bit 0 is the control bit and bit 1 is the target bit. As defined earlier in Eqn. (34), the R_y gate is responsible for specifying the angle of rotation for each qubit, and we have 10 angle parameters per layer because we have 10 qubits. The qubits are then entangled through 10 CNOT gates to achieve the entangled states. We can repeat this process as long as the end-to-end quantum computation is within the coherence time limit of our quantum computer. Because the coherence time is extremely limited today, we want to minimize the number of layers, l, to the smallest possible.

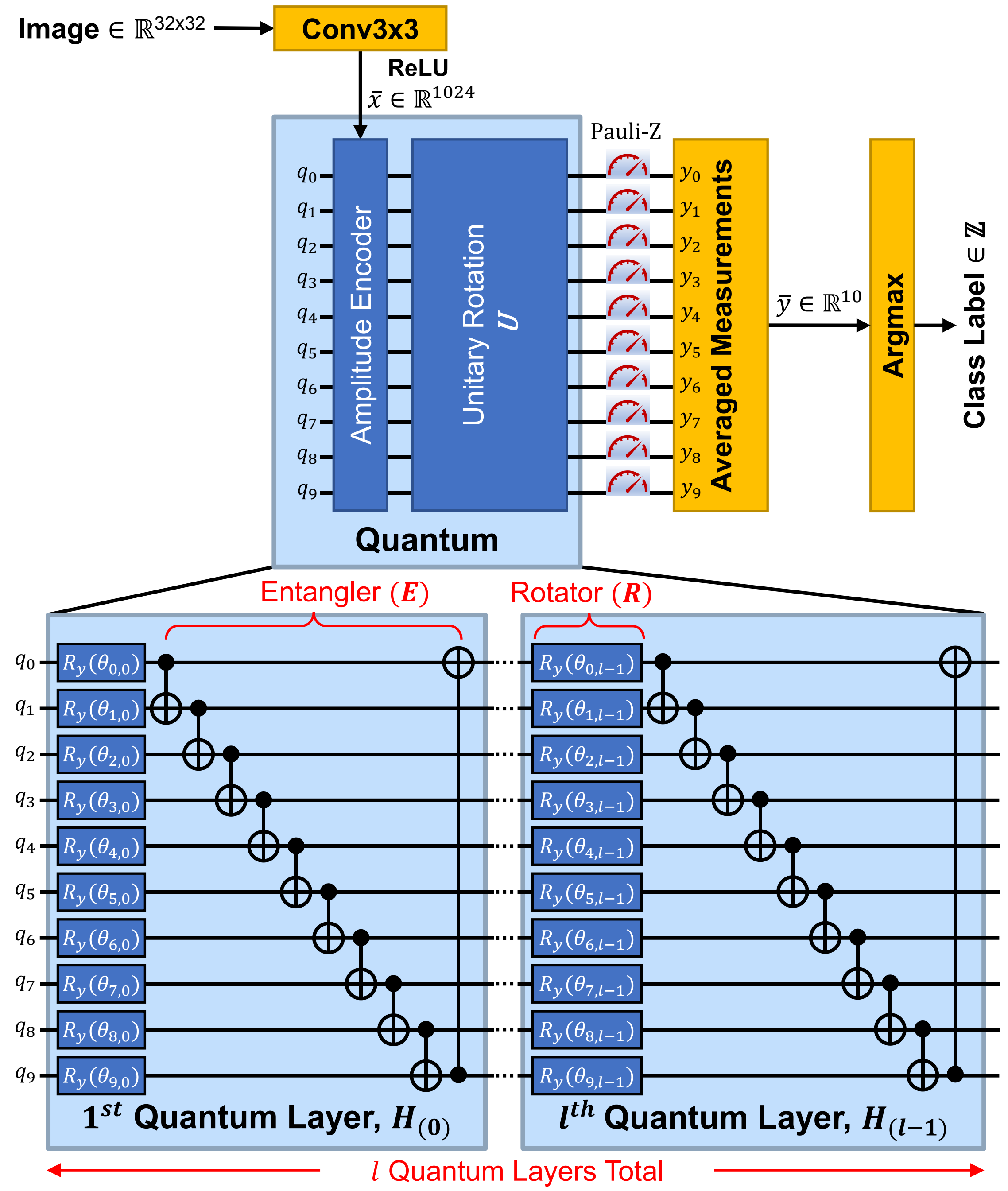

Image $\in \mathbb{R}^{32x32}$
Conv3x3
ReLU
$\bar{x} \in \mathbb{R}^{1024}$
Amplitude Encoder
Unitary Rotation U
Quantum
Pauli-Z
q_0 … q_9
y_0 … y_9
Averaged Measurements
$\bar{y} \in \mathbb{R}^{10}$
Argmax
Class Label $\in \mathbb{Z}$
Entangler ($\boldsymbol{E}$)
Rotator ($\boldsymbol{R}$)
$R_y(\theta_{0,0})$ … $R_y(\theta_{9,0})$
$R_y(\theta_{0,l-1})$ … $R_y(\theta_{9,l-1})$
1st Quantum Layer, $\boldsymbol{H}_{(0)}$
lth Quantum Layer, $\boldsymbol{H}_{(l-1)}$
l Quantum Layers Total

Fig. 13. Our hybrid quantum neural network architecture. An input image is first preprocessed using a 3 x 3 convolutional filter and ReLU in a classical computer (top). The amplitude encoder encodes the activations ($\bar{x}$) into the coefficients for the multi-qubit basis states of the quantum computer (middle). The unitary rotation ($\boldsymbol{U}$) has k

layers, and each layer has a rotator (10 R_y gates) and an entangler (10 CNOT gates). Quantum signals are measured with the Pauli-Z operator. The average values of the measurements ($\bar{y}$) are used for further classical postprocessing (right).

B. <u>Quantum Simulations</u>

We do not use any existing quantum computing framework for our simulations. Because the expected values of our hybrid QNN over many trials can be described analytically by a transfer function, it is the most efficient from a simulation point of view to derive the end-to-end analytical expression. We program the transfer function in a tensor computation package called PyTorch.[14] In our custom simulator, the amplitude encoder is simply a normalization step guaranteeing the amplitude vector to have a unit Euclidean norm. $\boldsymbol{U}$ has l layers, in which l is a design parameter. To derive an expression for $\boldsymbol{U}$, we break it down to multiple matrix multiplications and tensor products.

Each rotator depicted in Fig. 13 can be represented by the tensor product of 10 R_y gates:

$$\boldsymbol{R} = \otimes_{i=0}^{9} \boldsymbol{R}_y(\theta_i), \tag{41}$$

where $\boldsymbol{R}_y(\theta)$ is defined previously in Eqn. (34) and see the explanation below for our tensor notation.[15] The entangler in each layer is made of a ring of 10 CNOT gates arranged in a close boundary condition. The transfer function of the entangler can be further broken down into the product of the individual CNOT gates:

[14] Open-source software available at https://pytorch.org/

[15] We can write out the tensor product explicitly as: $\otimes_{i=0}^{9} R_y(\theta_\mathrm{i}) = R_y(\theta_0) \otimes R_y(\theta_1) \otimes R_y(\theta_2) \otimes R_y(\theta_3) \otimes R_y(\theta_4) \otimes R_y(\theta_5) \otimes R_y(\theta_6) \otimes R_y(\theta_7) \otimes R_y(\theta_8) \otimes R_y(\theta_9)$

$$\boldsymbol{E} = \prod_{i=0}^{9} \boldsymbol{E}_i, \tag{42}$$

where $\boldsymbol{E}_i$ defines the linear transform of a CNOT gate between bit i for the control bit and bit i+1 (modulo 10) as the target bit. More specifically, $\boldsymbol{E}_i$ has the following form:

$$\boldsymbol{E}_i = \begin{cases} \boldsymbol{I}_{2^{8-i}} \otimes \boldsymbol{CNOT} \otimes \boldsymbol{I}_{2^i}, \; if\; i < 9 \\ \begin{bmatrix} \boldsymbol{I}_{512} & 0 \\ 0 & \boldsymbol{I}_{256} \otimes \boldsymbol{NOT} \end{bmatrix}, \; if\; i = 9 \end{cases}, \tag{43}$$

where $\boldsymbol{I}_n$ is the n x n identity matrix. $\boldsymbol{CNOT}$ is defined in Eqn. (40), and $\boldsymbol{NOT}$ is defined as:

$$\boldsymbol{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{44}$$

In addition, the transfer function for a single quantum layer (***H***) in Fig. 13 (bottom) is defined as:

$$\boldsymbol{H}_{(j)} = \boldsymbol{R}_{(j)}\boldsymbol{E}, \tag{45}$$

where j is the layer index, $\boldsymbol{R}_{(j)}$ is the programmable rotator for each layer j from Eqn. (41), and ***E*** is the entangler from Eqn. (42). Finally, we arrive at the complete analytical expression for the rotation matrix ***U*** by multiplying the transfer function of all k layers:

$$\boldsymbol{U} = \prod_{j=0}^{l-1} \boldsymbol{H}_{(j)} = \prod_{j=0}^{l-1} \boldsymbol{R}_{(j)}\boldsymbol{E}. \tag{46}$$

The physical meaning of ***U*** is the effective transfer function implemented with the programmable quantum gates shown in Fig. 13. ***E*** is identical for all layers, and it contains no trainable parameters. By design, the number of qubits (n_c) in our system is equivalent to the number of class labels in the classification task.

C. Dataset Characteristics

We use the Modified National Institute of Standards and Technology hand-written digits dataset (MNIST) as an example classification problem [95]. Each data point in MNIST is a 28 x 28 pixels grayscale image of a single hand-written digit. It contains ten different classes, one for each of the numerical numbers 0 through 9. Overall, there are 60,000 training samples and 10,000 test samples. We preprocess the images by upscaling them to 32 x 32 pixels with linear interpolations. The resulting number of total pixels in the image (1024) is a power of two; this is for the ease of programming them into qubits.

D. Training Details

To train the proposed hybrid QNN, we measure the deviation from the desired outcome by using the cross-entropy loss as defined in Eqn. (13) between the network's output and the true label. The number of classes n_c in Eqn. (13) is also the number of qubits. We use end-to-end backpropagation to train both the convolution filter and the unitary rotation. Our complete hybrid QNN has the following chosen trainable parameters: the 3 x 3 convolution filter has 9 scalar weight parameters with 3 biases. For the quantum neural network, we have l x n_c (i.e., the number of layers x the number of classes) angles for the R_y gates in the quantum unitary rotation ($\boldsymbol{U}$).

We renormalize $\bar{y}$ through the softmax function defined in Eqn. (12) to obtain the probabilities of the data belonging to a particular class collectively represented by $\bar{p}$. Renormalizing $\bar{y}$ with softmax is optional in classification because the following argmax function picks the element with the highest value regardless of normalization. In other words, since softmax is a monotonic function, it does not change which vector element is the largest.

Nevertheless, it is nice to compute $\bar{p}$ because each of its vector elements represents the probability of being in a particular class.

We select a batch size of 64 and 20 training epochs. Our backpropagation uses the stochastic gradient descent algorithm with a learning rate (i.e. step size) of 0.003 and a momentum (i.e. effective mass) of 0.9, which adjust the weights toward the direction of the maximal gradient in the loss function with artificial momentum calculated to help the algorithm escape local minima. With the number of classes and qubits fixed (n_c = 10), we vary l (the number of layers) from 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, to 1024. For scientific rigor, we repeat each simulation four times with random initialization on the angles, namely, θ_i in Eqn. (41), by randomly sampling from a uniform distribution between $-\pi$ and π. The training set is shuffled randomly before each simulation.

4.4. Results & Discussion

A. The Intrinsic Dimensionality of a Dataset

The *PCA accuracy*, a metric to quantify the quality of signal reconstruction, is computed as follows: We first record the neural activations before the amplitude encoder in Fig. 13 and compress the recorded signals by using only the top k most significant eigenvectors (ranked by their eigenvalues). Then, we measure the average reconstruction error (e) by measuring the Euclidean distance between the reconstructed signals and the original ones. With 1024 eigenvectors, we can perfectly reconstruct the signals because that is the total dimensionality of the signal. With 1024 eigenvectors, the reconstruction error is zero, and the PCA accuracy is 1. In the case of reconstruction using a single eigenvector, the reconstruction error is a large value (e_{max}), and we define its PCA accuracy to be 0. Additionally, we define the PCA accuracy (η) for any reconstruction error (e) between 0 and e_{max} as:

$$\eta = PCA\ accuracy \triangleq 1 - e/e_{max} = f(k). \tag{47}$$

η is a function of k, the number of PCA components we used for the signal reconstruction. It is not possible to derive an analytic equation for f as it depends on the specific dataset we use; however, the relationship between η and k can be measured experimentally as to be shown in the next subsection.

Additionally, for linking η into our theory, we need to introduce the concept of *intrinsic dimensionality* for a dataset, denoted as d_{min}. For instance, when we take pictures of real-world objects to construct an image dataset, the camera we use to take these images can have an arbitrarily high resolution. Logically, the minimum amount of information we need to represent various objects in a computer should not depend on the size of the sensor; rather, there should be a minimal set of basis vectors required to represent the objects canonically. We define the size of this set as the intrinsic dimensionality of a dataset (d_{min}). There is no standard way to calculate the intrinsic dimensionality of the dataset d_{min}; the only criterion for the minimal set of basis vectors is that they must reconstruct the signals "well enough," which we define here as the PCA accuracy (η) exceeding 95%. Thus, d_{min} is the intrinsic dimensionality of the dataset defined as follows:

$$d_{min} \triangleq number\ of\ PCA\ components\ needed\ for\ \eta\ to\ be\ 0.95. \tag{48}$$

B. <u>Predicting the Required Quantum Depth</u>

Intuitively, the classification accuracy (ξ) for our hybrid QNN should increase with more quantum layers because, as the number of programmable angles increases, the network can rotate the dataset in more directions to achieve a better fit between the data and the decision boundaries. However, increasing the number of quantum layers (l) beyond the intrinsic dimensionality of the dataset (d_{min}) would yield a marginal improvement in the classification accuracy (ξ). The reason is that if most of the information about the dataset can be represented using d_{min} basis vectors, providing the additional capacity to rotate the data in directions orthogonal to the space spanned by them is not necessary. The main theory that we would like to verify with quantum simulation is that the unitary rotation ($\boldsymbol{U}$) requires at least d_{min} quantum layers to achieve a reasonable classification accuracy (ξ). This statement assumes that the number of classes in the classification problem, the number of qubits in the quantum computer, and the number of R_y rotation gates in a quantum layer are all the same (i.e., n_c = 10 as shown in Fig. 13). We will show our experimental data next.

After training using the procedure described previously in Sect. 4.3D, we use our hybrid QNN to classify the images from MNIST handwritten digit dataset into 10 categories (i.e., 0-9). See Sect. 4.3C for details about this dataset. The vertical bars in Fig. 14 show the classification accuracy (ξ) of our hybrid QNN as we vary l, the number of quantum layers. The dashed curve on the top plots the PCA reconstruction accuracy (η) achieved with the corresponding number of principal components (k). We measure the accuracy using the test set, which is the portion of the dataset that we set aside for model evaluation. In Fig. 14, there is a clear correlation between the PCA reconstruction accuracy and the QNN classification accuracy (ξ) when we align l and k on the same x-axis. In the case of PCA reconstruction, k is the number of eigenvectors we use to reconstruct the signals. In the case of the QNN, l is the number of quantum layers in $\boldsymbol{U}$. Our theory in Sect. 4.2C predicts that a QNN with l quantum layers can freely rotate k principal components, assuming each quantum layer has n_c tunable parameters. To elaborate, the number of parameters in our QNN is l x n_c, and it matches the degree of freedom k x n_c we need to solve a classification problem with n_c classes. Consequently, we see the two trends overlap in the figure.

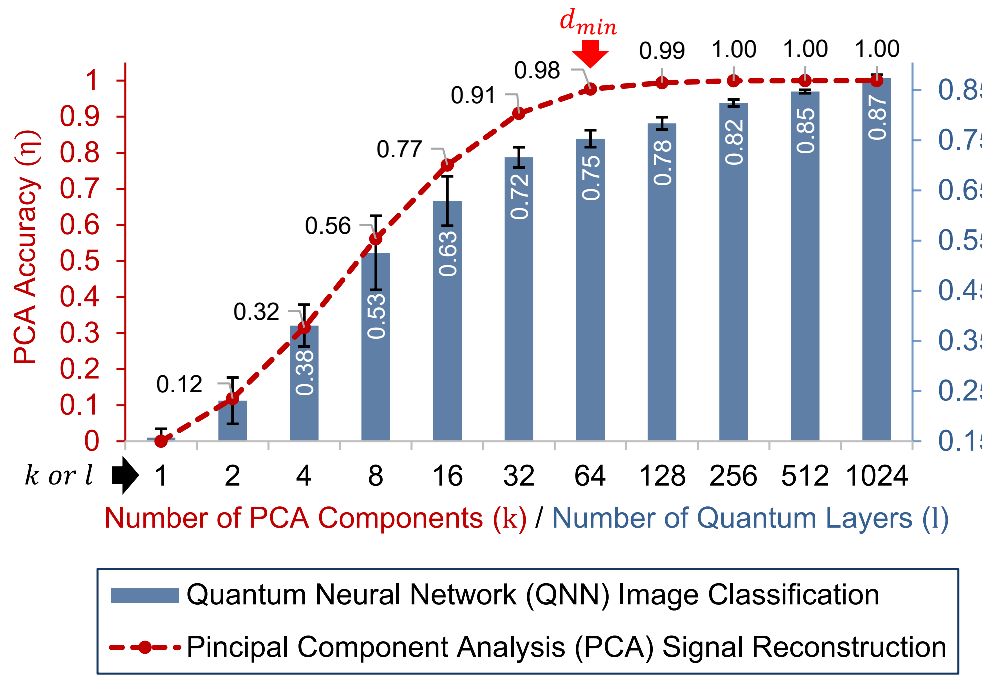

Fig. 14. Relationship between the intrinsic dataset dimensionality and the required quantum depth. Using the MNIST hand-written digit dataset, we perform principal component analysis (PCA) on the activation map ($\bar{x} \in \mathbb{R}^{1024}$) after the convolution and ReLU layer in Fig. 13. The weights for the convolution filters and the quantum rotation ($\boldsymbol{U}$) are trained end-to-end with backpropagation using the cross-entropy loss on the MNIST dataset. As defined in Eqn. (47), the PCA accuracy (η) quantifies the quality of the reconstructed signal (left vertical axis) using k PCA components (horizontal axis). On the right vertical axis, the QNN accuracy (ξ) measures the percentage of correctly predicted samples in the test set using l quantum layers (horizontal axis) in the unitary rotation ($\boldsymbol{U}$). As shown in the figure, η (dashed line) correlates with ξ (vertical bars). We define the intrinsic dimensionality of the dataset (d_{min}) as the number of PCA components needed for η to exceed 0.95, which is indicated with an arrow in the figure. The number

of quantum layers must match or exceed the d_{min} to achieve a reasonable classification accuracy ($\xi = 0.75$ in this example).

With this definition, we find d_{min} to be 64 using the dashed curve in Fig. 14 with the simulation stated before. This critical point is determined by the covariance of the pixels in the dataset, and it will impact our network design a great deal. Both the PCA accuracy (η) and the classification accuracy (ξ) rise rapidly for $1 < k < 64$, but the improvements slow down drastically afterward. The classification accuracy (ξ) increases by 75% with the first 64 quantum layers, but it only improves by 12% when l increases further from 64 to 1024 layers and plateaus at 87%. It takes a great deal of more eigenvectors to capture the last 5% of the variance because the PCA eigenvalues often have a long tail distribution. This is because the dimensionality of the sensor used to capture the image (i.e., the resolution of the camera, 32 x 32 for the example shown in Fig. 13) is often much higher than the space needed to represent the objects in the image; therefore, it becomes uneconomical for the number of quantum layers (l) to greatly exceeds d_{min}. This information is essential in the NISQ era because we often need to reduce the number of quantum layers to accommodate the highly constrained coherence time in today's quantum computers. As a first pass, practitioners can use the PCA accuracy curve (η) to set a corresponding number of quantum layers (l) equal to the intrinsic dimensionality (d_{min}) that will capture 95% of the variance information (e.g., $l = k = d_{min} = 64$ in our example), ensuring a satisfactory classification accuracy (e.g., $\xi = 0.75$ in our case). Additionally, using the PCA accuracy (η) to approximate the return on investment with additional quantum layers is an excellent rule of thumb for practical applications in the NISQ era.

4.5. Conclusion

In this study, we have proposed a novel hybrid QNN that can support multiclass classification problems. We use the hand-written number recognition as an example to show an accuracy of 87% on the MNIST dataset with full image resolution. In addition to demonstrating the possibility of using the unitary rotation to create a classification model in quantum computing, we study the decision boundaries drawn by our quantum system. As a crucial topic for classification models, decision boundaries specify the demarcation of data points in the input vector space for predicting their class labels. By deriving the equations for the decision boundaries of our hybrid QNN, we offer new insights into its decision process: the quantum gates rotate the data encoded in the multi-qubit basis states and move them into one of the cone-shaped decision regions. This interpretation provides a foundation for understanding the decision processes in QNNs for classification.

In addition, our key contribution is establishing a rule of thumb for determining the optimal (or minimal) number of quantum layers (d_{min}) to put in the network by performing the principal value analysis on the dataset. The unitary rotation requires a minimum number, d_{min} x n_c, of R_y gates to implement, where d_{min} is also the number of PCA components needed to properly reconstruct the dataset to 95% accuracy. We also show that there is a diminishing return of investment when we increase the number of quantum layers (l) beyond d_{min}. Machine learning practitioners can use our framework to balance and optimize the classification accuracy, the number of quantum gates, and the coherence time required to solve a particular classification problem.

Chapter 5 — Closing Remarks

Research never ends.

5.1. Summary

As we have demonstrated in the previous chapters, unitary neural networks are faster, more robust, and ready for quantum. Their speed improvement comes from the fact that unitary rotations preserved the Euclidean metric norm, and thus, it maintains the neural signal strengths. Although nonlinear activation functions might weaken the neural activations across multiple layers, the unitary constraint alone is sufficient to mitigate the exploding and diminishing gradient problem for a 143-layer network without normalization as we have shown in Chapter 2. The elimination of normalization increases the inference speed by up to 32% in an example of image recognition with our UniResNet architecture (Sect. 2.4), which is a novel neural architecture that uses unitary weights to stabilize the activations in a convolutional neural network. By using the Stiefel manifold, we invent a new way to generalize the concept of unitarity to rectangular weight matrices that are prevalent in most convolution filters; furthermore, the tunable parameters can be reduced by up to 50% due to the unitary constraints (Sect. 2.2).

Unitary neural nets are more robust because unitary transformation preserves the distance between two vectors. This unique property makes them resilient to adversarial attacks that are contrived with small perturbations on the input signals. By restraining the perturbations across the network via unitary constraints and by increasing the decision margin with the multi-margin loss, we can improve the prediction accuracy under adversarial attacks by up to an order of magnitude (Sect. 3.4). More specifically, in Chapter 3, we propose an enhanced neural architecture named UniBERT, which is a variant of the state-of-the-art Bidirectional Encoder Representations from Transformers architecture for natural language processing. Our UniBERT can deter adversarial attacks much better than the state-of-the-art defense methods based on data argumentation and regularization, outperforming the state-of-the-art methods by at least 15% in post-attack classification accuracies across tasks and attack recipes under the toughest evaluation metrics. Chapter 2 and Chapter 3 demonstrate that unitary neural nets are efficient, robust, and fundamentally stable—an excellent architectural choice for all neural networks when implemented in classical computing.

Unitary neural nets provide an effortless transition to quantum information processing. The gates in quantum computers perform unitary transformations; as a result, the weights in a unitary neural net can be decomposed into a cascade of rotation gates in the quantum domain (Sect. 4.3). The exponential number (2^n) of quantum states generated by n qubits provides an opportunity for initiating a new technology curve, a quantum version of Moore's law. We show in Sect. 4.4 that the required number of quantum gates depends on the principal value decomposition of the dataset; our framework and treatise provide a practical guide to estimating the number of qubits required, and this result gives the design requirement for a quantum computer's coherence time to solve classification and other classically hard problems.

Lastly, we would like to add a discussion on the overall view of this dissertation. We began with a comprehensive study of the performance gains in the classical domain to show that unitary neural networks outperform the state-of-the-art in inference speed and adversarial robustness. Our work here offers a practical methodology that makes neural computation more efficient and safer for a wide range of industrial applications, including mission-critical computer vision and natural language processing. Moreover, we champion unitary neural networks as a stepping stone to quantum computing because there is a direct translation between the classical unitary weights and quantum gates. This unique property shortens the design cycle of quantum neural nets via straightforward conversion between simulation and hardware deployment.

5.2. Outlook

Many open questions remain in the treatment of unitary neural networks. In the classical domain, unitary constraints can be implemented in various ways, including matrix exponentiation (Sect. 2.2A) and QR decomposition (Sect. 3.3B). In this treatise, our experience shows that QR decomposition is faster than matrix exponentiation on graphics computing units. Nevertheless, there are also iterative methods such as calculating the weight matrices' deviations from the unitary group and making small corrections at each training step, which may have negligible training time penalties and could be more computationally efficient than our current techniques for implementing the unitary constraints.

In the quantum domain, it is essential to build a nonlinear quantum computer for machine learning. Deep learning relies on multiple layers of nonlinearity to create intricate decision boundaries in the feature space. The lack of nonlinear functions in quantum computing severely handicaps the usefulness of quantum neural nets. Any quantum system designed to obey the linear Schrödinger equation will not support the nonlinear transformation of the quantum state amplitudes. A common workaround in today's literature is to use measurement as a source of nonlinearity or perform nonlinear processing in a classical computer. Either way, we lose the benefit of the exponential number of states generated by qubits once the wavefunctions collapse. We predict that the next quantum leap will come from the invention of nonlinear quantum gates.

To implement nonlinear gates, researchers need to not only characterize nonlinear quantum devices but develop a mathematical framework to model them. One related field of research is nonlinear optics. For instance, the Kerr nonlinearity describes the nonlinear polarization for a dielectric material under an external electric field, resulting in a nonlinear

Schrödinger equation for the wavefunctions [96]. This nonlinear effect is used to build nonlinear components in optical quantum computers [97]. On the other hand, there is a wide research gap in building and modeling nonlinear gates for superconducting qubits.

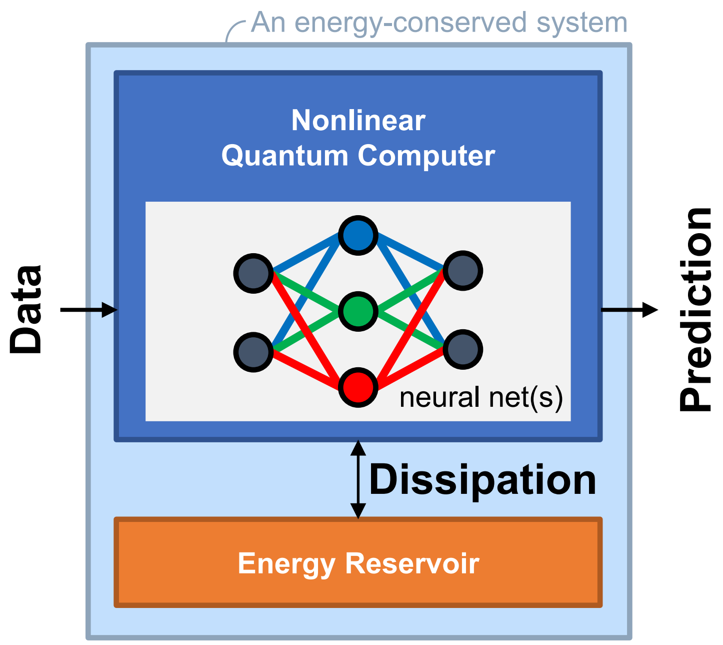

Fig. 15. Nonlinear quantum neural networks are realized by dissipative quantum computing. A key limitation of today's quantum computers is that it is completely linear by design; as a result, quantum machine learning requires tedious workarounds for implementing the nonlinearities required to achieve the desired model capacity. We propose a paradigm shift in quantum research to incorporate dissipation as a source of nonlinearity for a machine learning model (such as the neural network depicted in the middle of this figure). The effect of dissipation can be quantified using the nonlinear Schrödinger equation and incorporated into neural network training. With the introduction of nonlinearity in the system, nonlinear quantum computers not only

alleviate the bottleneck in scaling the number of qubits but also open up a wide range of quantum algorithms to make quantum computing a step closer to general computing.

Another interesting research direction for creating nonlinearity is to use decoherence as a source of nonlinearity [98]. If we can analytically express the environmental coupling and model the dissipation as an integral part of network training, we may be able to circumvent the problem of the coherence time while supporting nonlinear transforms, achieving exponential scaling at a discounted rate. Nonlinear quantum computers such as the one depicted in Fig. 15 kill two birds with one stone: Instead of fighting the quantum decoherence, express it as the nonlinearity for a machine learning model (e.g., a neural network) by studying the nonlinear Schrödinger equation associated with the dissipation phenomenon [99], [100]. Such a futuristic machine learning system will be a mixture of the linear, non-dissipative, unitary layers and the nonlinear, dissipative, non-unitary layers, providing a quantum leap toward a new paradigm for computing and information processing.

References

[1] H.-Y. Chang and K. L. Wang, “Deep Unitary Convolutional Neural Networks,” in *Artificial Neural Networks and Machine Learning – ICANN 2021*, Cham, 2021, pp. 170–181. doi: 10.1007/978-3-030-86340-1_14.

[2] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974, doi: 10.1109/JSSC.1974.1050511.

[3] M. Bohr, “A 30 Year Retrospective on Dennard’s MOSFET Scaling Paper,” *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007, doi: 10.1109/N-SSC.2007.4785534.

[4] G. E. Moore, “Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.,” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, Sep. 2006, doi: 10.1109/N-SSC.2006.4785860.

[5] C. M. Christensen, *The innovator’s dilemma: when new technologies cause great firms to fail*. Boston, Massachusetts: Harvard Business Review Press, 2000.

[6] A. Messiah, *Quantum Mechanics*. Courier Corporation, 2014.

[7] V. B. Rojansky, *Introductory Quantum Mechanics*. Prentice-Hall, Incorporated, 1938.

[8] E. Mollick, “Establishing Moore’s Law,” *IEEE Annals of the History of Computing*, vol. 28, no. 3, pp. 62–75, Jul. 2006, doi: 10.1109/MAHC.2006.45.

[9] Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Dec. 2015, pp. 770–778. Accessed: Sep. 02, 2020. [Online]. Available: http://arxiv.org/abs/1512.03385

[11] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, “On the Number of Linear Regions of Deep Neural Networks,” *Advances in Neural Information Processing Systems*, vol. 27, 2014, Accessed: Feb. 22, 2021. [Online]. Available: https://papers.nips.cc/paper/2014/hash/109d2dd3608f669ca17920c511c2a41e-Abstract.html

[12] I. Goodfellow, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[13] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training Recurrent Neural Networks,” *arXiv:1211.5063 [cs]*, Feb. 2013, Accessed: Feb. 22, 2021. [Online]. Available: http://arxiv.org/abs/1211.5063

[14] E. Haber and L. Ruthotto, “Stable architectures for deep neural networks,” *Inverse Problems*, vol. 34, no. 1, p. 014004, Dec. 2017, doi: 10.1088/1361-6420/aa9a90.

[15] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Madison, WI, USA, Jun. 2010, pp. 807–814.

[16] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: 10.1109/72.279181.

[17] Sergey Ioffe and Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *Proceedings of the 32nd International Conference on Machine Learning*, Jun. 2015, vol. 37, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[18] J. Wei, “Forget the Learning Rate, Decay Loss,” *IJMLC*, vol. 9, no. 3, pp. 267–272, Jun. 2019, doi: 10.18178/ijmlc.2019.9.3.797.

[19] M. Lezcano-Casado and D. Martínez-Rubio, “Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group,” in *Proceedings of the 36th International Conference on Machine Learning*, Jun. 2019, vol. 97, pp. 3794–3803. [Online]. Available: http://proceedings.mlr.press/v97/lezcano-casado19a.html

[20] M. Arjovsky, A. Shah, and Y. Bengio, “Unitary Evolution Recurrent Neural Networks,” in *International Conference on Machine Learning*, Jun. 2016, pp. 1120–1128. Accessed: Feb. 22, 2021. [Online]. Available: http://proceedings.mlr.press/v48/arjovsky16.html

[21] K. Helfrich, D. Willmott, and Q. Ye, “Orthogonal Recurrent Neural Networks with Scaled Cayley Transform,” in *International Conference on Machine Learning*, Jul. 2018, pp. 1969–1978. Accessed: Feb. 22, 2021. [Online]. Available: http://proceedings.mlr.press/v80/helfrich18a.html

[22] L. Jing *et al.*, “Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs,” in *International Conference on Machine Learning*, Jul. 2017, pp. 1733–1741. Accessed: Feb. 22, 2021. [Online]. Available: http://proceedings.mlr.press/v70/jing17a.html

[23] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey, “Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections,” in *International Conference on Machine Learning*, Jul. 2017, pp. 2401–2409. Accessed: Feb. 22, 2021. [Online]. Available: http://proceedings.mlr.press/v70/mhammedi17a.html

[24] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, “Full-Capacity Unitary Recurrent Neural Networks,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4880–4888. Accessed: Aug. 21, 2020. [Online]. Available: http://papers.nips.cc/paper/6327-full-capacity-unitary-recurrent-neural-networks.pdf

[25] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li, “Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks,” presented at the Thirty-Second AAAI Conference on Artificial Intelligence, 2017. Accessed: Aug. 21, 2020. [Online]. Available: http://arxiv.org/abs/1709.06079

[26] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, “On orthogonality and learning recurrent networks with long term dependencies,” in *International Conference on Machine Learning*, Jul. 2017, pp. 3570–3578. Accessed: Feb. 22, 2021. [Online]. Available: http://proceedings.mlr.press/v70/vorontsov17a.html

[27] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu, “Orthogonal Convolutional Neural Networks,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 11502–11512. doi: 10.1109/CVPR42600.2020.01152.

[28] R. Gilmore and R. Hermann, *Lie Groups, Lie Algebras, and Some of Their Applications*. New York: John Wiley & Sons, 1974.

[29] R. Gilmore, *Lie Groups, Physics, and Geometry: An Introduction for Physicists, Engineers and Chemists*. New York: Cambridge University Press, 2008.

[30] P. Bader, S. Blanes, and F. Casas, “Computing the Matrix Exponential with an Optimized Taylor Polynomial Approximation,” *Mathematics*, vol. 7, no. 12, Art. no. 12, Dec. 2019, doi: 10.3390/math7121174.

[31] Torch Contributors, “torch.matrix_exp — PyTorch 1.7.0 documentation,” 2019. https://pytorch.org/docs/stable/generated/torch.matrix_exp.html (accessed Nov. 24, 2020).

[32] A. Paszke *et al.*, “Automatic differentiation in PyTorch,” *NIPS 2017 Workshop Autodiff Submission*, Oct. 2017, doi: https://openreview.net/pdf?id=BJJsrmfCZ.

[33] A. Edelman, T. A. Arias, and S. T. Smith, “The Geometry of Algorithms with Orthogonality Constraints,” *SIAM J. Matrix Anal. & Appl.*, vol. 20, no. 2, pp. 303–353, Jan. 1998, doi: 10.1137/S0895479895290954.

[34] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, “A Survey of Accelerator Architectures for Deep Neural Networks,” *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020, doi: 10.1016/j.eng.2020.01.007.

[35] A. Araujo, B. Negrevergne, Y. Chevaleyre, and J. Atif, “On Lipschitz Regularization of Convolutional Layers using Toeplitz Matrix Theory,” Vancouver, Canada, Feb. 2021.

Accessed: Mar. 03, 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03107420

[36] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” University of Toronto, 2009. Accessed: Feb. 03, 2021. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[37] V. Chiley *et al.*, “Online Normalization for Training Neural Networks,” in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8433–8443. Accessed: Nov. 03, 2020. [Online]. Available: http://papers.nips.cc/paper/9051-online-normalization-for-training-neural-networks.pdf

[38] Y. Wu and K. He, “Group Normalization,” *Int J Comput Vis*, vol. 128, no. 3, pp. 742–755, Mar. 2020, doi: 10.1007/s11263-019-01198-w.

[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization,” *arXiv:1607.06450 [cs, stat]*, Jul. 2016, Accessed: Jan. 13, 2021. [Online]. Available: http://arxiv.org/abs/1607.06450

[40] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance Normalization: The Missing Ingredient for Fast Stylization,” *arXiv:1607.08022 [cs]*, Nov. 2017, Accessed: Jan. 13, 2021. [Online]. Available: http://arxiv.org/abs/1607.08022

[41] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[42] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative Study of CNN and RNN for Natural Language Processing,” *arXiv:1702.01923 [cs]*, Feb. 2017, Accessed: Nov. 14, 2021. [Online]. Available: http://arxiv.org/abs/1702.01923

[43] A. Miaschi and F. Dell’Orletta, “Contextual and Non-Contextual Word Embeddings: an in-depth Linguistic Investigation,” in *Proceedings of the 5th Workshop on Representation Learning for NLP*, Online, Jul. 2020, pp. 110–119. doi: 10.18653/v1/2020.repl4nlp-1.15.

[44] D. Hendrycks and K. Gimpel, “Gaussian Error Linear Units (GELUs),” arXiv, arXiv:1606.08415, Jul. 2020. doi: 10.48550/arXiv.1606.08415.

[45] J. Salazar, D. Liang, T. Q. Nguyen, and K. Kirchhoff, “Masked Language Model Scoring,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, Jul. 2020, pp. 2699–2712. doi: 10.18653/v1/2020.acl-main.240.

[46] N. Akhtar and A. Mian, “Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey,” *IEEE Access*, vol. 6, pp. 14410–14430, 2018, doi: 10.1109/ACCESS.2018.2807385.

[47] H. Karimi, T. Derr, and J. Tang, "Characterizing the Decision Boundary of Deep Neural Networks," *arXiv:1912.11460 [cs, stat]*, Jun. 2020, Accessed: Nov. 14, 2021. [Online]. Available: http://arxiv.org/abs/1912.11460

[48] E. Wong and Z. Kolter, "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope," in *Proceedings of the 35th International Conference on Machine Learning*, Jul. 2018, pp. 5286–5295. Accessed: Oct. 22, 2021. [Online]. Available: https://proceedings.mlr.press/v80/wong18a.html

[49] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep Text Classification Can be Fooled," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, Jul. 2018, pp. 4208–4215. doi: 10.24963/ijcai.2018/585.

[50] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment," *arXiv:1907.11932 [cs]*, Apr. 2020, Accessed: Aug. 11, 2021. [Online]. Available: http://arxiv.org/abs/1907.11932

[51] S. Ren, Y. Deng, K. He, and W. Che, "Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, Jul. 2019, pp. 1085–1097. doi: 10.18653/v1/P19-1103.

[52] Y. Xie, Z. Gu, X. Fu, L. Wang, W. Han, and Y. Wang, "Misleading Sentiment Analysis: Generating Adversarial Texts by the Ensemble Word Addition Algorithm," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, Nov. 2020, pp. 590–596. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00103.

[53] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "TextBugger: Generating Adversarial Text Against Real-world Applications," presented at the Network and Distributed System Security Symposium, San Diego, CA, 2019. doi: 10.14722/ndss.2019.23138.

[54] C. Si *et al.*, "Better Robustness by More Coverage: Adversarial and Mixup Data Augmentation for Robust Finetuning," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, Online, Aug. 2021, pp. 1569–1576. doi: 10.18653/v1/2021.findings-acl.137.

[55] J. Zhao, P. Wei, and W. Mao, "Robust Neural Text Classification and Entailment via Mixup Regularized Adversarial Training," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, Jul. 2021, pp. 1778–1782. doi: 10.1145/3404835.3463122.

[56] B. Wang *et al.*, "InfoBERT: Improving Robustness of Language Models from An Information Theoretic Perspective," *arXiv:2010.02329 [cs]*, Mar. 2021, Accessed: Sep. 27, 2021. [Online]. Available: http://arxiv.org/abs/2010.02329

[57] S. Rosset, J. Zhu, and T. Hastie, “Margin Maximizing Loss Functions,” in *Advances in Neural Information Processing Systems*, 2004, vol. 16. Accessed: Nov. 14, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2003/hash/0fe473396242072e84af286632d3f0ff-Abstract.html

[58] G. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, “Large Margin Deep Networks for Classification,” in *Advances in Neural Information Processing Systems*, 2018, vol. 31. Accessed: Nov. 04, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/42998cf32d552343bc8e460416382dca-Abstract.html

[59] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep Hypersphere Embedding for Face Recognition,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, Jul. 2017, pp. 6738–6746. doi: 10.1109/CVPR.2017.713.

[60] E. Romero, L. Marquez, and X. Carreras, “Margin maximization with feed-forward neural networks: a comparative study with SVM and AdaBoost.” 2004.

[61] F. Wang, J. Cheng, W. Liu, and H. Liu, “Additive Margin Softmax for Face Verification,” *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, Jul. 2018, doi: 10.1109/LSP.2018.2822810.

[62] N. Bansal, X. Chen, and Z. Wang, “Can We Gain More from Orthogonality Regularizations in Training Deep CNNs?,” *arXiv:1810.09102 [cs, stat]*, Oct. 2018, Accessed: May 05, 2021. [Online]. Available: http://arxiv.org/abs/1810.09102

[63] C. Xu, X. Li, and M. Yang, “An Orthogonal Classifier for Improving the Adversarial Robustness of Neural Networks,” *arXiv:2105.09109 [cs]*, Sep. 2021, Accessed: Jan. 13, 2022. [Online]. Available: http://arxiv.org/abs/2105.09109

[64] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Neural Photo Editing with Introspective Adversarial Networks,” *arXiv:1609.07093 [cs, stat]*, Feb. 2017, Accessed: May 05, 2021. [Online]. Available: http://arxiv.org/abs/1609.07093

[65] G. H. Golub and C. F. Van Loan, *Matrix computations*, Fourth edition. Baltimore: The Johns Hopkins University Press, 2013.

[66] Y. Zhu *et al.*, “Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 19–27. doi: 10.1109/ICCV.2015.11.

[67] X. Zhang, J. Zhao, and Y. LeCun, “Character-level Convolutional Networks for Text Classification,” in *Advances in Neural Information Processing Systems*, 2015, vol. 28. Accessed: Nov. 14, 2021. [Online]. Available:

https://proceedings.neurips.cc/paper/2015/hash/250cf8b51c773f3f8dc8b4be867a9a02-Abstract.html

[68] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, “A large annotated corpus for learning natural language inference,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, Sep. 2015, pp. 632–642. doi: 10.18653/v1/D15-1075.

[69] T. Wolf *et al.*, “HuggingFace’s Transformers: State-of-the-art Natural Language Processing,” *arXiv:1910.03771 [cs]*, Jul. 2020, Accessed: Oct. 06, 2020. [Online]. Available: http://arxiv.org/abs/1910.03771

[70] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: May 09, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980

[71] N. Mrkšić *et al.*, “Counter-fitting Word Vectors to Linguistic Constraints,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, Jun. 2016, pp. 142–148. doi: 10.18653/v1/N16-1018.

[72] G. A. Miller, “WordNet: a lexical database for English,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995, doi: 10.1145/219717.219748.

[73] D. Cer *et al.*, “Universal Sentence Encoder for English,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Brussels, Belgium, Nov. 2018, pp. 169–174. doi: 10.18653/v1/D18-2029.

[74] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, “TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, Oct. 2020, pp. 119–126. doi: 10.18653/v1/2020.emnlp-demos.16.

[75] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *arXiv:1907.11692 [cs]*, Jul. 2019, Accessed: Jan. 01, 2022. [Online]. Available: http://arxiv.org/abs/1907.11692

[76] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,” *arXiv:1909.11942 [cs]*, Feb. 2020, Accessed: Oct. 14, 2020. [Online]. Available: http://arxiv.org/abs/1909.11942

[77] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv:1910.01108 [cs]*, Feb. 2020, Accessed: Jan. 16, 2022. [Online]. Available: http://arxiv.org/abs/1910.01108

[78] P. A. Gargini, “How to successfully overcome inflection points, or long live Moore’s law,” *Computing in Science Engineering*, vol. 19, no. 2, pp. 51–62, Mar. 2017, doi: 10.1109/MCSE.2017.32.

[79] M. T. Bohr and I. A. Young, “CMOS Scaling Trends and Beyond,” *IEEE Micro*, vol. 37, no. 6, pp. 20–29, Nov. 2017, doi: 10.1109/MM.2017.4241347.

[80] S. K. Saha, “Modeling Process Variability in Scaled CMOS Technology,” *IEEE Design Test of Computers*, vol. 27, no. 2, pp. 8–16, Mar. 2010, doi: 10.1109/MDT.2010.50.

[81] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, 10th anniversary ed. Cambridge ; New York: Cambridge University Press, 2010.

[82] S. Imre and F. Balázs, *Quantum computing and communications : an engineering approach*. Chichester, West Sussex, England ; Wiley, 2005.

[83] A. Streltsov, G. Adesso, and M. B. Plenio, “Colloquium: Quantum coherence as a resource,” *Rev. Mod. Phys.*, vol. 89, no. 4, p. 041003, Oct. 2017, doi: 10.1103/RevModPhys.89.041003.

[84] R. Youssef, “Measuring and Simulating T_1 and T_2 for Qubits,” Fermi National Accelerator Lab. (FNAL), Batavia, IL (United States), FERMILAB-POSTER-20-115-SCD, Aug. 2020. doi: 10.2172/1656632.

[85] “Cramming More Power Into a Quantum Device,” *IBM Research Blog*, Mar. 04, 2019. https://www.ibm.com/blogs/research/2019/03/power-quantum-device/ (accessed Apr. 19, 2022).

[86] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018, doi: 10.22331/q-2018-08-06-79.

[87] V. Havlíček *et al.*, “Supervised learning with quantum-enhanced feature spaces,” *Nature*, vol. 567, no. 7747, pp. 209–212, Mar. 2019, doi: 10.1038/s41586-019-0980-2.

[88] W. Jiang, J. Xiong, and Y. Shi, “A co-design framework of neural networks and quantum circuits towards quantum advantage,” *Nat Commun*, vol. 12, no. 1, Art. no. 1, Jan. 2021, doi: 10.1038/s41467-020-20729-5.

[89] W. Jiang, J. Xiong, and Y. Shi, “When Machine Learning Meets Quantum Computers: A Case Study,” in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, New York, NY, USA, Jan. 2021, pp. 593–598. doi: 10.1145/3394885.3431629.

[90] C.-C. Chen, M. Watabe, K. Shiba, M. Sogabe, K. Sakamoto, and T. Sogabe, “On the Expressibility and Overfitting of Quantum Circuit Learning,” *ACM Transactions on Quantum Computing*, vol. 2, no. 2, pp. 1–24, Jul. 2021, doi: 10.1145/3466797.

[91] D. McMahon, *Quantum computing explained*. Hoboken, N.J: Wiley-Interscience, 2008.

[92] S. B. Sontz, “Spin and SU(2),” in *An Introductory Path to Quantum Theory: Using Mathematics to Understand the Ideas of Physics*, S. B. Sontz, Ed. Cham: Springer International Publishing, 2020, pp. 149–163. doi: 10.1007/978-3-030-40767-4_14.

[93] S. Gong, V. N. Boddeti, and A. K. Jain, “On the Intrinsic Dimensionality of Image Representations,” 2019, pp. 3987–3996. Accessed: Apr. 19, 2022. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/html/Gong_On_the_Intrinsic_Dimensionality_of_Image_Representations_CVPR_2019_paper.html

[94] R. Shepard, S. R. Brozell, and G. Gidofalvi, “The Representation and Parametrization of Orthogonal Matrices,” *J. Phys. Chem. A*, vol. 119, no. 28, pp. 7924–7939, Jul. 2015, doi: 10.1021/acs.jpca.5b02015.

[95] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[96] G. Fibich, *The nonlinear Schrödinger equation : singular solutions and optical collapse*. Cham [Switzerland] ; Springer, 2015.

[97] R. Yanagimoto, T. Onodera, E. Ng, L. G. Wright, P. L. McMahon, and H. Mabuchi, “Engineering a Kerr-based Deterministic Cubic Phase Gate via Gaussian Operations,” *Phys. Rev. Lett.*, vol. 124, no. 24, p. 240503, Jun. 2020, doi: 10.1103/PhysRevLett.124.240503.

[98] G. K. Sadiek, “Decoherence in nonlinear quantum systems,” Ph.D., Purdue University, United States -- Indiana. Accessed: Apr. 26, 2022. [Online]. Available: https://www.proquest.com/docview/305498863/abstract/174E41D2FA494DC3PQ/1

[99] H.-P. Breuer and F. Petruccione, *The theory of open quantum systems*. Oxford ; New York: Oxford University Press, 2002.

[100] G. Schaller, *Open Quantum Systems Far from Equilibrium*, vol. 881. Cham: Springer International Publishing, 2014. doi: 10.1007/978-3-319-03877-3.