mathematics

MDPI

Article

The Representation Theory of Neural Networks

Marco Armenta [1,2,*] and Pierre-Marc Jodoin [2]

[1] Department of Mathematics, Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada
[2] Department of Computer Science, Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada; pierre-marc.jodoin@usherbrooke.ca
* Correspondence: marco.armenta@usherbrooke.ca

Abstract: In this work, we show that neural networks can be represented via the mathematical theory of quiver representations. More specifically, we prove that a neural network is a quiver representation with activation functions, a mathematical object that we represent using a *network quiver*. Furthermore, we show that network quivers gently adapt to common neural network concepts such as fully connected layers, convolution operations, residual connections, batch normalization, pooling operations and even randomly wired neural networks. We show that this mathematical representation is by no means an approximation of what neural networks are as it exactly matches reality. This interpretation is algebraic and can be studied with algebraic methods. We also provide a quiver representation model to understand how a neural network creates representations from the data. We show that a neural network saves the data as quiver representations, and maps it to a geometrical space called the *moduli space*, which is given in terms of the underlying oriented graph of the network, i.e., its *quiver*. This results as a consequence of our defined objects and of understanding how the neural network computes a prediction in a combinatorial and algebraic way. Overall, representing neural networks through the quiver representation theory leads to 9 consequences and 4 inquiries for future research that we believe are of great interest to better understand what neural networks are and how they work.

Keywords: neural networks; quiver representations; data representations

Citation: Armenta, M.; Jodoin, P.-M. The Representation Theory of Neural Networks. *Mathematics* **2021**, *9*, 3216. https://doi.org/10.3390/math9243216

Academic Editors: Marina Alexandra Pedro Andrade and Maria Alves Teodoro

Received: 30 October 2021
Accepted: 3 December 2021
Published: 13 December 2021

1. Introduction

Neural networks have achieved unprecedented performances in almost every area where machine learning is applicable [1–3]. Throughout its history, computer science has had several turning points with groundbreaking consequences that unleashed the power of neural networks. To name a few, one might regard the chain rule backpropagation [4], the invention of convolutional layers [5] and recurrent models [4], the advent of low-cost specialized parallel hardware (mostly GPUs) [6] and the exponential growth of available training data as some of the most important factors behind today's success of neural networks.

Ironically, despite our understanding of every atomic element of a neural network and our capability to successfully train it, it is still difficult with today's formalism to understand *what* makes neural networks so effective. As neural nets increase in size, the combinatorics between its weights and activation functions makes it impossible (at least today) to formally answer questions such as: (i) why neural networks (almost) always converge towards a global minima regardless of their initialization, the data it is trained on and the associated loss function; (ii) what is the true capacity of a neural net? (iii) what are the true generalization capabilities of a neural net?

One may hypothesize that the limited understanding of these fundamental concepts derives from the more or less formal representation that we have of these machines. Since the 1980s, neural nets have been mostly represented in two ways: (i) a cascade of non-linear atomic operations (be it, a series of neurons with their activation functions, layers,

convolution blocks, etc.) often represented graphically (e.g., Figure 3 by He et al. [7]) and (ii) a point in an N dimensional Euclidean space (where N is the number of weights in the network) lying on the slope of a loss landscape that an optimizer ought to climb down [8].

In this work, we propose a fundamentally different way to represent neural networks. Based on quiver representation theory, we provide a new mathematical footing to represent neural networks as well as the data they process. We show that this mathematical representation is by no means an approximation of what neural networks are as it tightly matches reality.

In this paper, we do not focus on how neural networks learn, but rather on the intrinsic properties of their architectures and their forward pass of data. Therefore providing new insights on how to understand neural networks. Our mathematical formulation accounts for the wide variety of architectures there are, and also usages and behaviors of today's neural networks. For this, we study the combinatorial and algebraic nature of neural networks by using ideas coming from the mathematical theory of quiver representations [9,10]. Although this paper focuses on feed-forward networks, a combinatorial argument on recurrent neural networks can be made to apply our results to them: the cycles in recurrent neural networks are only applied a finite number of times, and once unraveled they combinatorially become networks that feed information in a single direction with shared weights [11].

This paper is based on two observations that expose the algebraic nature of neural networks and how it is related to quiver representations:

1. When computing a prediction, neural networks are quiver representations together with activation functions.
2. The forward pass of data through the network is encoded as quiver representations.

Everything else in this work is a mathematical consequence of these two observations. Our main contributions can be summarized by the following six items:

1. We provide the first explicit link between representations of quivers and neural networks.
2. We show that quiver representations gently adapt to common neural network concepts such as fully connected layers, convolution operations, residual connections, batch normalization, pooling operations, and any feed-forward architecture, since this is a universal description of neural networks.
3. We prove that algebraic isomorphisms of neural networks preserve the network function and obtain, as a corollary, that ReLU networks are positive scale invariant [12–14].
4. We present the theoretical interpretation of data in terms of the architecture of the neural network and of quiver representations.
5. We mathematically formalize a modified version of the manifold hypothesis [3,11] in terms of the combinatorial architecture of the network.
6. We provide constructions and results supporting existing intuitions in deep learning while discarding others, and bring new concepts to the table.

2. Previous Work

In the theoretical description of the deep neural optimization paradigm given by Choromanska et al. [15], the authors underline that "*clearly the model (neural net) contains several dependencies as one input is associated with many paths in the network. That poses a major theoretical problem in analyzing these models as it is unclear how to account for these dependencies*". Interestingly, this is exactly what quiver representations are about [9,10,16].

While as far as we know, quiver representation theory has never been used to study neural networks, some authors have nonetheless used a subset of it, sometimes unbeknownst to them. It is the case of the so-called *positive scale invariance* of ReLU networks which Dinh et al. [12] used to mathematically prove that most notions of loss flatness cannot be used directly to explain generalization. This property of ReLU networks has also been used by Neyshabur et al. [14] to improve the optimization of ReLU networks. In their paper, they propose the *Path-SGD* (stochastic gradient descent), which is an approximate gradient descent method with respect to a path-wise regularizer. Furthermore, Meng et al. [13] defined a space where points are ReLU networks with the same network function, which

they use to find better gradient descent paths. In this paper (cf. Theorem 1 and Corollary 2), we prove that positive scale invariance of ReLU networks is a property derived from the representation theory of neural networks that we present in the following sections. We interpret these results as evidence of the algebraic nature of neural networks, as they exactly match the basic definitions of representation theory (i.e., quiver representations and morphisms of quiver representations).

Wood and Shawe-Taylor [17] used group representation theory to account for symmetries in the layers of a neural network. Our mathematical approach is different since quiver representations are representations of algebras [9] and not of groups. Besides, Wood and Shawe-Taylor [17] present architectures that match mathematical objects with nice properties while we define the objects that model the computations of the neural network. We prove that quiver representations are more suited to study networks due to their combinatorial and algebraic nature.

Healy and Caudell [18] mathematically represent neural networks by objects called *categories*. However, as mentioned by the authors, their representation is an approximation of what neural nets are as they do not account for each of their atomic elements. In contrast, our quiver representation approach includes every computation involved in a neural network, be it a neural operation (i.e., dot product + activation function), layer operations (fully connected, convolutional, pooling) as well as batch normalization. As such, our representation is a universal description of neural networks, i.e., the results and consequences of this paper apply to all neural networks.

Quiver representations have been used to find lower-dimensional sub-space structures of datasets [19] without, however, any relation to neural networks. Our interpretation of data is orthogonal to this one since we look at how neural networks interpret the data in terms of every single computation they perform.

Following the discussion by S. Arora in his 2018 ICML tutorial [20] on the characteristics of a theory for deep learning, our goal is precisely this. Namely, to provide a theoretical footing that can validate and formalize certain intuitions about deep neural nets and lead to new insights and new concepts. One such intuition is related to feature map visualization. It is well known that feature maps can be visualized into images showing the input signal characteristics and thus providing intuitions on the behavior of the network and its impact on an image [21,22]. This notion is strongly supported by our findings. Namely, our data representation introduced in Section 6 is a thin quiver representation that contains the network features (i.e., neuron outputs or feature maps) induced by the data. Said otherwise, our data representation includes both the network structure and the neuron's inputs and outputs induced by a forward pass of a single data sample (see Equation (5) in page 19 and the proof of Theorem 2). Our data quiver representations contain every feature map during a forward pass of data and so it is aligned with the notion of *representations* in representation learning [3,11,23].

We show in Section 7 that our data representations lie into a so-called *moduli space*. Interestingly, the dimension of the moduli space is the same value that was computed by Zheng et al. [24] and used to measure the capacity of ReLU networks. They empirically confirmed that the dimension of the moduli space is directly linked to generalization. Our results suggest that the findings mentioned above can be generalized to any neural network via representation theory.

The moduli space also formalizes a modified version of the manifold hypothesis for the data see [3] (Chapter 5.11.3). This hypothesis states that high-dimensional data (typically images and text) live on a thin and yet convoluted manifold in their original space. We show that this data manifold can be mapped to the moduli space while carrying the feature maps induced by the data, and then it is related to notions appearing in manifold learning [11,23]. Our results, therefore, create a new bridge between the mathematical study of these moduli spaces [25–27] and the study of the training dynamics of neural networks inside these moduli spaces.

Naive pruning of neural networks [28] where the smallest weights get pruned is also explained by our interpretation of the data and the moduli space (see consequence 4 on Section 7.1.2), since the coordinates of the data quiver representations inside the moduli space are given as a function of the weights of the network and the activation outputs of each neuron on a forward pass (cf. Equation (5) in page 19).

There exist empirical results where, up to certain restrictions, the activation functions can be learned [29] and our interpretation of the data supports why this is a good idea in terms of the moduli space. For further details see Section 7.2.3.

3. Preliminaries of Quiver Representations

Before we show how neural networks are related to quiver representations, we start by defining the basic concepts of quiver representation theory [9,10,16]. The reader can find a glossary with all the definitions introduced in this and the next chapters at the end of this paper.

Definition 1 ([9] (Chapter 2)). *A **quiver** Q is given by a tuple $(\mathcal{V}, \mathcal{E}, s, t)$ where $(\mathcal{V}, \mathcal{E})$ is an oriented graph with a set of vertices $\mathcal{V}$ and a set of oriented edges $\mathcal{E}$, and maps $s, t : \mathcal{E} \to \mathcal{V}$ that send $\epsilon \in \mathcal{E}$ to its source vertex $s(\epsilon) \in \mathcal{V}$ and target vertex $t(\epsilon) \in \mathcal{V}$, respectively.*

Throughout the present paper, we work only with quivers whose sets of edges and vertices are finite.

Definition 2 ([9] (Chapter 2)). *A **source vertex** of a quiver Q is a vertex $v \in \mathcal{V}$ such that there are no oriented edges $\epsilon \in \mathcal{E}$ with target $t(\epsilon) = v$. A **sink vertex** of a quiver Q is a vertex $v \in \mathcal{V}$ such that there are no oriented edges $\epsilon \in \mathcal{E}$ with source $s(\epsilon) = v$. A **loop** in a quiver Q is an oriented edge ϵ such that $s(\epsilon) = t(\epsilon)$.*

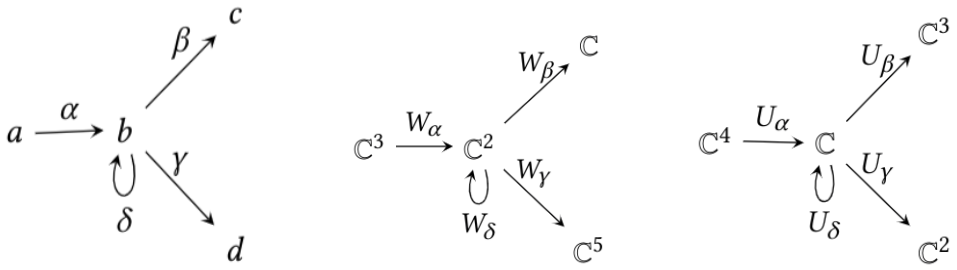
Definition 3 ([9] (Chapter 3)). *If Q is a quiver, a **quiver representation** of Q is given by a pair of sets*

$$W := \left((W_v)_{v \in \mathcal{V}}, (W_\epsilon)_{\epsilon \in \mathcal{E}}\right)$$

where the W_v's are vector spaces indexed by the vertices of Q, and the W_ϵ's are linear maps indexed by the oriented edges of Q, such that for every edge $\epsilon \in \mathcal{E}$

$$W_\epsilon : W_{s(\epsilon)} \to W_{t(\epsilon)}.$$

Figure 1a illustrates a quiver Q while Figure 1b,c are two quiver representations of Q.



(a) **(b)** **(c)**

Figure 1. (**a**) A quiver Q with vertices $\mathcal{V} = \{a, b, c, d\}$ and oriented edges $\mathcal{E} = \{\alpha, \beta, \gamma, \delta\}$, where the source and target maps are defined by $s(\alpha) = a$, $s(\beta) = b$, $s(\gamma) = b$, $s(\delta) = b$, $t(\alpha) = b$, $t(\beta) = c$, $t(\gamma) = d$ and $t(\delta) = b$. (**b**) A quiver representation W over Q, where vertices a to d are complex 3D, 2D, 1D and 5D vector spaces, and W_α is a 2×3 matrix, W_β is a 1×2 matrix, W_γ is a 5×2 matrix and W_δ is a 2×2 matrix. (**c**) Another quiver representation U over Q, where a to d are complex 4D, 1D, 3D and 2D vector spaces, and U_α is a 1×4 matrix, U_β is a 3×1 matrix, U_γ is a 2×1 matrix and U_δ is a 1×1 matrix.

Definition 4 ([9] (Chapter 3)). *Let Q be a quiver and let W and U be two representations of Q. A **morphism of representations** $\tau : W \to U$ is a set of linear maps $\tau = (\tau_v)_{v \in \mathcal{V}}$ indexed by the vertices of Q, where $\tau_v : W_v \to U_v$ is a linear map such that $\tau_{t(\epsilon)} W_\epsilon = U_\epsilon \tau_{s(\epsilon)}$ for every $\epsilon \in \mathcal{E}$.*

To illustrate this definition, one may consider the quiver Q and its representations W and U of Figure 1. The morphism between W and U via the linear maps τ are pictured in Figure 2a. As shown, each τ_v is a matrix which allows to transform the vector space of vertex v of W into the vector space of vertex v of U.

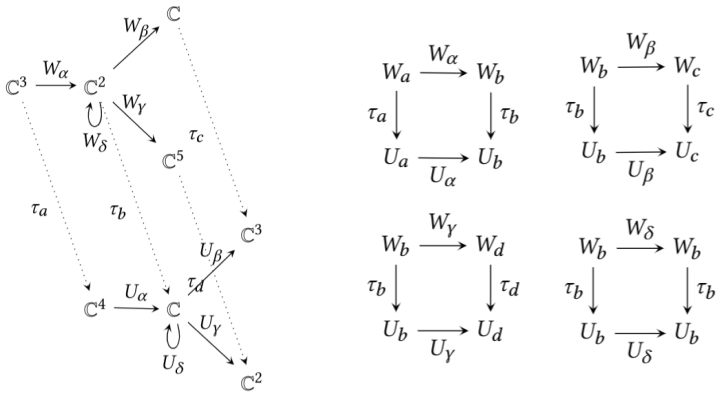


Figure 2. (**a**) A morphism of representations $\tau : W \to U$ is given by a family of matrices $\tau = (\tau_v)_{v \in \mathcal{V}}$, such that $\tau_a : \mathbb{C}^3 \to \mathbb{C}^4$, $\tau_b : \mathbb{C}^2 \to \mathbb{C}$, $\tau_c : \mathbb{C} \to \mathbb{C}^3$ and $\tau_d : \mathbb{C}^5 \to \mathbb{C}^2$ satisfy that $\tau_b W_\alpha = U_\alpha \tau_a$, $\tau_c W_\beta = U_\beta \tau_b$, $\tau_d W_\gamma = U_\gamma \tau_b$, $\tau_b W_\delta = U_\delta \tau_b$. (**b**) Four diagrams showing that the transformations τ_v must make them commutative for $\tau : W \to U$ to be a morphism of representations.

Definition 5. *Let Q be a quiver and let W and U be two representations of Q. If there is a morphism of representations $\tau : W \to U$ where each τ_v is an invertible linear map, then W and U are said to be **isomorphic representations**.*

The previous definition is equivalent to the usual categorical definition of isomorphism, see [9] (Chapter 3). Namely, a morphism of representations $\tau : W \to U$ is an isomorphism if there exists a morphism of representations $\eta : U \to W$ such that $\eta \circ \tau = id_W$ and $\tau \circ \eta = id_U$. Observe here that the composition of morphisms is defined as a coordinate-wise composition, indexed by the vertices of the quiver.

In Section 4, we will be working with a particular type of quiver representations, where the vector space of each vertex is in 1D. These 1D representations are called thin representations, and the morphisms of representations between thin representations are easily described.

Definition 6. *A **thin representation** of a quiver Q is a quiver representation W such that $W_v = \mathbb{C}$ for all $v \in V$.*

If W is a thin representation of Q, then every linear map W_ϵ is a 1×1 matrix, so W_ϵ is given by multiplication with a fixed complex number. We may and will identify every linear map between one-dimensional spaces with the number whose multiplication defines it.

Before we move on to neural networks, we will introduce the notion of group and action of a group.

Definition 7 ([30] (Chapter 1)). *A non-empty set G is called a **group** if there exists a function $\cdot : G \times G \to G$, called the product of the group denoted $a \cdot b$, such that*

- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ *for all* $a, b, c \in G$.

- *There exists an element $e \in G$ such that $e \cdot a = a \cdot e = a$ for all $a \in G$, called the **identity** of G.*
- *For each $a \in G$ there exists $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.*

For example, the set of non-zero complex numbers $\mathbb{C}^*$ (and also the non-zero real numbers $\mathbb{R}^*$) with the usual multiplication operation forms a group. Usually, one does not write the product of the group as a dot and just concatenates the elements to denote multiplication $ab = a \cdot b$, as for the product of numbers.

Definition 8 ([30] (Chapter 3))**.** *Let G be a group and let X be a set. We say that there is an **action of** G **on** X if there exists a map $\cdot : G \times X \to X$ such that*

- *$e \cdot x = x$ for all $x \in X$, where $e \in G$ is the identity.*
- *$a \cdot (b \cdot x) = (ab) \cdot x$, for all $a, b \in G$ and all $x \in X$.*

In our case, G will be a group indexed by the vertices of Q, and the set X will be the set of thin quiver representations of Q.

Let W be a thin representation of a quiver Q. Given a choice of invertible (non-zero) linear maps $\tau_v : \mathbb{C} \to \mathbb{C}$ for every $v \in \mathcal{V}$, we are going to construct a thin representation U such that $\tau = (\tau_v)_{v \in \mathcal{V}} : W \to U$ is an isomorphism of representations. Since U is thin, we have that $U_v = \mathbb{C}$ for all $v \in \mathcal{V}$. Let $\epsilon : a \to b$ be an edge of $\mathcal{E}$, we define the group action as follows,

$$U_\epsilon = W_\epsilon \cdot \frac{\tau_b}{\tau_a}. \tag{1}$$

Thus, for every edge $\epsilon \in \mathcal{E}$ we get a commutative diagram

$$\begin{array}{ccc} W_{s(\epsilon)} & \xrightarrow{W_\epsilon} & W_{t(\epsilon)} \\ {\scriptstyle \tau_{s(\epsilon)}}\downarrow & & \downarrow{\scriptstyle \tau_{t(\epsilon)}} \\ U_{s(\epsilon)} & \xrightarrow[U_\epsilon]{} & U_{t(\epsilon)}. \end{array}$$

The construction of the thin representation U from the thin representation W and the choice of invertible linear maps τ, defines an action on thin representations of a group. The set of all possible isomorphisms $\tau = (\tau_v)_{v \in \mathcal{V}}$ of thin representations of Q forms such a group.

Definition 9. *The **change of basis group** of thin representations over a quiver Q is*

$$G = \prod_{v \in \mathcal{V}} \mathbb{C}^*,$$

where $\mathbb{C}^*$ denotes the multiplicative group of non-zero complex numbers. That is, the elements of G are vectors of non-zero complex numbers $\tau = (\tau_1, \ldots, \tau_n)$ indexed by the set $\mathcal{V}$ of vertices of Q, and the group operation between two elements $\tau = (\tau_1, \ldots, \tau_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_n)$ is by definition

$$\tau\sigma := (\tau_1\sigma_1, \ldots, \tau_n\sigma_n).$$

We use the action notation for the action of the group G on thin representations. Namely, for $\tau \in G$ of the form $\tau = (\tau_v)_{v \in \mathcal{V}}$ and a thin representation W of Q, the thin representation U constructed above is denoted $\tau \cdot W$.

4. Neural Networks

In this section, we connect the dots between neural networks and the basic definitions of quiver representation theory that we presented before. However, before we do so, let us mention that since the vector space of each vertex of a quiver representation is

defined over the complex numbers, it implies that the weights on the neural networks that we are to present will also be complex numbers. Despite some papers on complex neural networks [31], this approach may seem unorthodox. However, the use of complex numbers is a mathematical pre-requisite for the upcoming notion of moduli space that we will introduce in Section 7. Observe also, that this does not mean that in practice neural networks should be based on complex numbers. It only means that neural networks in practice, which are based upon real numbers, trivially satisfy the condition of being complex neural networks, and therefore the mathematics derived from using complex numbers apply to neural networks over real numbers.

For the rest of this paper, we will focus on a special type of quiver Q that we call *network quiver*. A network quiver Q has no oriented cycles other than loops. Moreover, a sub-set of d source vertices of Q are called the **input vertices**. The source vertices that are not input vertices are called **bias vertices**. Let k be the number of all sinks of Q, we call these the **output vertices**. All other vertices of Q are called **hidden vertices**.

Definition 10. *A quiver Q is **arranged by layers** if it can be drawn from left to right arranging its vertices in columns such that:*

- *There are no oriented edges from vertices on the right to vertices on the left.*
- *There are no oriented edges between vertices in the same column, other than loops and edges from bias vertices.*

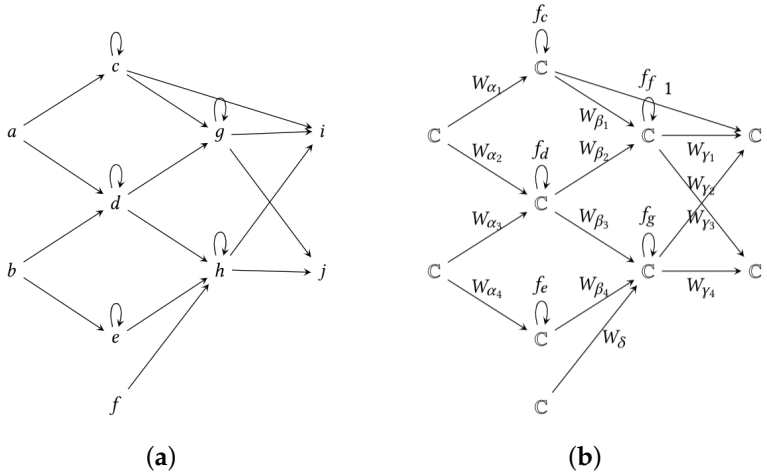
*The first layer on the left, called the **input layer**, will be formed by the d input vertices. The last layer on the right, called the **output layer**, will be formed by the k output vertices. The layers that are not input nor output layers are called **hidden layers**. We enumerate the hidden layers from left to right as 1st hidden layer, 2nd hidden layer, 3rd hidden layer, and so on.*

From now on Q will always denote a quiver with d input vertices and k output vertices.

Definition 11. *A **network quiver** Q is a quiver arranged by layers such that:*

1. *There are no loops on source (i.e., input and bias) nor sink vertices;*
2. *There is exactly one loop on each hidden vertex.*

An example of a network quiver can be found in Figure 3a.



(**a**) (**b**)

Figure 3. (**a**) A network quiver Q whose input layer is given by the vertices a and b, the vertex f is a bias vertex and there is a skip connection from vertex c to vertex i. Note that we did not label the edges to lighten the diagram. (**b**) A neural network over Q where $W_{\alpha_1}, W_{\alpha_2} \ldots, W_\delta$ are linear maps given by multiplication with a number, and the functions $f = (f_c, f_d, f_e, f_f, f_g)$ are the activation functions (could be sigmoid, tanh, ReLU, ELU, etc.)

Definition 12. *The **delooped** quiver Q° of Q is the quiver obtained by removing all loops of Q. We denote $Q^\circ = (\mathcal{V}, \mathcal{E}^\circ, s^\circ, t^\circ)$.*

When a neural network computes a forward pass (be it a multilayer perceptron, a convolutional neural network and even a randomly wired neural network [32]), the weight between two neurons is used to multiply the output signal of the first neuron and the result is fed to the second neuron. Since multiplying with a number (the weight) is a linear map, we get that a weight is used as a linear map between two 1D vector spaces during inference. Therefore the weights of a neural network define a thin quiver representation of the delooped quiver Q° of its network quiver Q, every time it computes a prediction.

When a neural network computes a forward pass, we get a combination of two things:

1. A thin quiver representation.
2. Activation functions.

Definition 13. *An **activation function** is a one variable non-linear function $f : \mathbb{C} \to \mathbb{C}$ differentiable except in a set of measure zero.*

Remark 1. *An activation function can, in principle, be linear. Nevertheless, neural network learning occurs with all its benefits only in the case where activation functions are fundamentally non-linear. Here, we want to provide a universal language for neural networks, so we will work with neural networks with non-linear activation functions, unless explicitly stated otherwise, for example as in our data representations in Section 6.*

We will encode the point-wise usage of activation functions as maps assigned to the loops of a network quiver.

Definition 14. *A **neural network** over a network quiver Q is a pair (W, f) where W is a thin representation of the delooped quiver Q° and $f = (f_v)_{v \in \mathcal{V}}$ are activation functions, assigned to the loops of Q.*

An example of neural network (W, f) over a network quiver Q can be seen in Figure 3b. The words **neuron** and **unit** refer to the combinatorics of a vertex together with its activation function in a neural network over a network quiver. The **weights** of a neural network (W, f) are the complex numbers defining the maps W_ϵ for all $\epsilon \in \mathcal{E}$.

When computing a prediction, we have to take into account two things:

- The activation function is applied to the sum of all input values of the neuron;
- The activation output of each vertex is multiplied by each weight going out of that neuron.

Once a network quiver and a neural network (W, f) are chosen, a decision has to be made on how to compute with the network. For example, a hidden neuron may compute an inner product of its inputs followed by the activation function, but others, like max-pooling, output the maximum of the input values. We account for this by specifying in the next definition how every type of vertex is used to compute.

Definition 15. *Let (W, f) be a neural network over a network quiver Q and let $x \in \mathbb{C}^d$ be an input vector of the network. Denote by ζ_v the set of edges of Q with target v. The **activation output of the vertex** $v \in \mathcal{V}$ **with respect to** x after applying a forward pass is denoted $\boldsymbol{a}(W, f)_v(x)$ and is computed as follows:*

- *If $v \in \mathcal{V}$ is an input vertex, then $\boldsymbol{a}(W, f)_v(x) = x_v$;*
- *If $v \in \mathcal{V}$ is a bias vertex, then $\boldsymbol{a}(W, f)_v(x) = 1$;*
- *If $v \in \mathcal{V}$ is a hidden vertex, then $\boldsymbol{a}(W, f)_v(x) = f_v\left(\sum_{\alpha \in \zeta_v} W_\alpha \boldsymbol{a}(W, f)_{s(\alpha)}(x)\right)$;*
- *If $v \in \mathcal{V}$ is an output vertex, then $\boldsymbol{a}(W, f)_v(x) = \sum_{\alpha \in \zeta_v} W_\alpha \boldsymbol{a}(W, f)_{s(\alpha)}(x)$;*

- *If $v \in \mathcal{V}$ is a max-pooling vertex, then $\boldsymbol{a}(W,f)_v(x) = \boldsymbol{max}_\alpha \, Re\big(W_\alpha \boldsymbol{a}(W,f)_{s(\alpha)}(x)\big)$, where Re denotes the real part of a complex number, and the maximum is taken over all $\alpha \in \mathcal{E}$ such that $t(\alpha) = v$.*

We will see in the next chapter how and why average pooling vertices do not require a different specification on the computation rule, because it can be written in terms of these same rules.

The previous definition is equivalent to the basic operations of a neural net, which are affine transformations followed by point-wise non-linear activation functions, see Appendix A where we clarify this with an example. The advantage of using the combinatorial expression of Definition 15 is twofold, (i) it allows to represent any architecture, even randomly wired neural networks [32], and (ii) it allows to simplify the notation on proofs concerning the network function.

For our purposes, it is convenient to consider no activation functions on the output vertices. This is consistent with current deep learning practices as one can consider the activation functions of the output neurons to be part of the loss function (like softmax + cross-entropy or as done by Dinh et al. [12]).

Definition 16. *Let (W,f) be a neural network over a network quiver Q. The **network function** of the neural network is the function*

$$\Psi(W,f) : \mathbb{C}^d \to \mathbb{C}^k$$

where the coordinates of $\Psi(W,f)(x)$ are the activation outputs of the output vertices of (W,f) (often called the "score" of the neural net) with respect to an input vector $x \in \mathbb{C}^d$.

The only difference in our approach is the combinatorial expression of Definition 15 which can be seen as a neuron-wise computation, that in practice is performed by layers for implementation purposes. These expressions will be useful to prove our more general results.

We now extend the notion of isomorphism of quiver representations to isomorphism of neural networks. For this, we have to take into account that isomorphisms of quiver representations carry the commutative diagram conditions given by all the edges in the quiver, as shown in Figure 2. For neural networks, the activation functions are non-linear, but this does not prevent us from putting a commutative diagram condition on activation functions as well. Therefore, an isomorphism of quiver representations acts on a neural network in the sense of the following definition.

Definition 17. *Let (W,f) and (V,g) be neural networks over the same network quiver Q. A **morphism of neural networks** $\tau : (W,f) \to (V,g)$ is a morphism of thin quiver representations $\tau : W \to V$ such that $\tau_v = 1$ for all $v \in \mathcal{V}$ that is not a hidden vertex, and for every hidden vertex $v \in \mathcal{V}$ the following diagram is commutative*

$$\begin{array}{ccc} \mathbb{C} & \xrightarrow{f_v} & \mathbb{C} \\ {\scriptstyle \tau_v}\downarrow & & \downarrow{\scriptstyle \tau_v} \\ \mathbb{C} & \xrightarrow[g_v]{} & \mathbb{C}. \end{array}$$

*A morphism of neural networks $\tau : (W,f) \to (V,g)$ is an **isomorphism of neural networks** if $\tau : W \to V$ is an isomorphism of quiver representations. We say that two neural networks over Q are **isomorphic** if there exists an isomorphism of neural networks between them.*

Remark 2. *The terms 'network morphism' [33], 'isomorphic neural network' and 'isomorphic network structures' [34,35] have already been used with different approaches. In this work, we will not refer to any of those terms.*

Definition 18. *The **hidden quiver** of Q, denoted by $\widetilde{Q} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{s}, \widetilde{t})$, is given by the hidden vertices $\widetilde{\mathcal{V}}$ of Q and all the oriented edges $\widetilde{\mathcal{E}}$ between hidden vertices of Q that are not loops.*

Said otherwise, $\widetilde{Q}$ is the same as the delooped quiver Q° but without the source and sink vertices.

Definition 19. *The **group of change of basis** for neural networks is denoted as*

$$\widetilde{G} = \prod_{v \in \widetilde{\mathcal{V}}} \mathbb{C}^*.$$

*An element of the change of basis group $\widetilde{G}$ is called a **change of basis** of the neural network (W, f).*

Note that this group has as many factors as hidden vertices of Q. Given an element $\widetilde{\tau} \in \widetilde{G}$ we can induce $\tau \in G$, where G is the change of basis group of thin representations over the delooped quiver Q°. We do this by assigning $\tau_v = 1$ for every $v \in \mathcal{V}$ that is not a hidden vertex. Therefore, we will simply write τ for elements of $\widetilde{G}$ considered as elements of G.

The action of the group $\widetilde{G}$ on a neural network (W, f) is defined on a given element $\tau \in \widetilde{G}$ and a neural network (W, f) by

$$\tau \cdot (W, f) = (\tau \cdot W, \tau \cdot f),$$

where $\tau \cdot W$ is the thin representation such that for each edge $\epsilon \in \mathcal{E}$, the linear map $(\tau \cdot W)_\epsilon = W_\epsilon \dfrac{\tau_{t(\epsilon)}}{\tau_{s(\epsilon)}}$ following the group action of Equation (1), and the activation $\tau \cdot f$ on the hidden vertex $v \in \mathcal{V}$ is given by

$$(\tau \cdot f)_v(x) = \tau_v f\left(\frac{x}{\tau_v}\right) \text{ for all } x \in \mathbb{C}. \tag{2}$$

Observe that $(\tau \cdot W, \tau \cdot f)$ is a neural network such that $\tau : (W, f) \to (\tau \cdot W, \tau \cdot f)$ is an isomorphism of neural networks. This leads us to the following theorem, which is an important corner stone of our paper. Please refer to Appendix A for an illustration of this proof.

Theorem 1. *If $\tau : (W, f) \to (V, g)$ is an isomorphism of neural networks, then $\Psi(W, f) = \Psi(V, g)$.*

Proof. Let $\tau : (W, f) \to (V, g)$ be an isomorphism of neural networks over Q and $\epsilon : s(\epsilon) \to t(\epsilon)$ an oriented edge of Q. Considering the group action of Equation (1), if $s(\epsilon)$ and $t(\epsilon)$ are hidden vertices then $V_\epsilon = W_\epsilon \cdot \dfrac{\tau_{t(\epsilon)}}{\tau_{s(\epsilon)}}$. However, if $s(\epsilon)$ is a source vertex, then $\tau_{s(\epsilon)} = 1$ and $V_\epsilon = W_\epsilon \tau_{t(\epsilon)}$. Additionally, if $t(\epsilon)$ is an output vertex, then $\tau_{t(\epsilon)} = 1$ and $V_\epsilon = \dfrac{W_\epsilon}{\tau_{s(\epsilon)}}$. Furthermore, for every hidden vertex $v \in \widetilde{\mathcal{V}}$ we get the activation function $g_v(z) = \tau_v \cdot f_v\left(\dfrac{z}{\tau_v}\right)$ for all $z \in \mathbb{C}$.

We proceed with a forward pass to compare the activation outputs of both neural networks with respect to the same input vector. Let $x \in \mathbb{C}^d$ be the input vector of the networks, for every source vertex $v \in \mathcal{V}$ we have

$$\mathbf{a}(W,f)_v(x) = \mathbf{a}(V,g)_v(x) = \begin{cases} x_v \in \mathbb{C} & \text{if } v \text{ is an input neuron,} \\ 1 & \text{if } v \text{ is a bias neuron.} \end{cases} \tag{3}$$

Now let $v \in \mathcal{V}$ be a vertex in the first hidden layer and ζ_v the set of edges between the source vertices and $v \in \mathcal{V}$, the activation output of v in (W,f) is

$$\mathbf{a}(W,f)_v(x) = f_v\left(\sum_{\epsilon\in\zeta_v} W_\epsilon \cdot \mathbf{a}(W,f)_{s(\epsilon)}(x)\right).$$

As an illustration, if (W,f) is the neural network of Figure 3, the source vertices would be a,b,f, the first hidden layer vertices would be c,d,e and the weights W_ϵ in the previous equation would be $\{W_{\alpha_2}, W_{\alpha_3}\}$ when $v = d$. We now calculate in (V,g) the activation output of the same vertex v,

$$\begin{aligned} \mathbf{a}(V,g)_v(x) &= \tau_v f_v\left(\frac{1}{\tau_v}\sum_{\epsilon\in\zeta_v} V_\epsilon \cdot \mathbf{a}(V,g)_{s(\epsilon)}(x)\right) \\ &= \tau_v f_v\left(\frac{1}{\tau_v}\sum_{\epsilon\in\zeta_v} W_\epsilon\tau_{t(\epsilon)} \cdot \mathbf{a}(V,g)_{s(\epsilon)}(x)\right) \end{aligned}$$

since $t(\epsilon) = v$, then $\tau_{t(\epsilon)} = \tau_v$ and

$$= \tau_v f_v\left(\sum_{\epsilon\in\zeta_v} W_\epsilon \cdot \mathbf{a}(V,g)_{s(\epsilon)}(x)\right)$$

and since $s(\epsilon)$ is a source vertex, it follows from Equation (3) that $\mathbf{a}(W,f)_{s(\epsilon)}(x) = \mathbf{a}(V,g)_{s(\epsilon)}(x)$ and

$$\begin{aligned} &= \tau_v f_v\left(\sum_{\epsilon\in\zeta_v} W_\epsilon \cdot \mathbf{a}(W,f)_{s(\epsilon)}(x)\right) \\ &= \tau_v \mathbf{a}(W,f)_v(x), \end{aligned}$$

Assume now that $v \in \mathcal{V}$ is in the second hidden layer (e.g., vertex g or h in Figure 3), the activation output of v in (V,g) is

$$\begin{aligned} \mathbf{a}(V,g)_v(x) &= \tau_v f_v\left(\frac{1}{\tau_v}\sum_{\epsilon\in\zeta_v} V_\epsilon \cdot \mathbf{a}(V,g)_{s(\epsilon)}(x)\right) \\ &= \tau_v f_v\left(\frac{1}{\tau_v}\sum_{\epsilon\in\zeta_v} \frac{W_\epsilon\tau_v}{\tau_{s(\epsilon)}}\mathbf{a}(V,g)_{s(\epsilon)}(x)\right) \\ &= \tau_v f_v\left(\sum_{\epsilon\in\zeta_v} \frac{W_\epsilon}{\tau_{s(\epsilon)}}\mathbf{a}(V,g)_{s(\epsilon)}(x)\right) \end{aligned}$$

and since $\mathbf{a}(V,g)_{s(\epsilon)}(x) = \tau_{s(\epsilon)}\mathbf{a}(W,f)_{s(\epsilon)}(x)$ from the equation above, then

$$\begin{aligned} &= \tau_v f_v\left(\sum_{\epsilon\in\zeta_v} \frac{W_\epsilon}{\tau_{s(\epsilon)}}\tau_{s(\epsilon)}\mathbf{a}(W,f)_{s(\epsilon)}(x)\right) \\ &= \tau_v f_v\left(\sum_{\epsilon\in\zeta_v} W_\epsilon\mathbf{a}(W,f)_{s(\epsilon)}(x)\right) \\ &= \tau_v \mathbf{a}(W,f)_v(x). \end{aligned}$$

Inductively, we obtain that $\mathbf{a}(V,g)_v(x) = \tau_v \mathbf{a}(W,f)_v(x)$ for every vertex $v \in \mathcal{V}$. Finally, the coordinates of $\Psi(W,f)(x)$ are the activation outputs of (W,f) on the output vertices, and analogously for $\Psi(V,g)(x)$. Since $\tau_v = 1$ for every output vertex $v \in \mathcal{V}$, we obtain

$$\Psi(W,f)(x) = \Psi(V,g)(x)$$

which proves that an isomorphism between two neural networks (W,f) and (V,g) preserves the network function. □

Remark 3. *Max-pooling represents a different operation to obtain the activation output of neurons. After applying an isomorphism τ to a neural network (W,f), where the vertex $v \in \mathcal{V}$ is a max-pooling vertex we obtain an isomorphic neural network (V,g), whose activation output on vertex v is given by the following formula:*

$$\boldsymbol{a}(V,g)_v(x) = \begin{cases} \boldsymbol{max}_{\substack{\alpha \in \mathcal{E} \\ t(\alpha) = v}} & Re\big(V_\alpha \boldsymbol{a}(V,g)_{s(\alpha)}(x)\big) \quad \text{if } Re(\tau_v) \geq 0 \\ \boldsymbol{min}_{\substack{\alpha \in \mathcal{E} \\ t(\alpha) = v}} & Re\big(V_\alpha \boldsymbol{a}(V,g)_{s(\alpha)}(x)\big) \quad \text{if } Re(\tau_v) < 0, \end{cases}$$

and

$$V_\alpha \boldsymbol{a}(V,g)_{s(\alpha)}(x) = \tau_{t(\alpha)} W_\alpha \tau_{s(\alpha)}^{-1} \tau_{s(\alpha)} \boldsymbol{a}(W,f)_{s(\alpha)}(x) = \tau_{t(\alpha)} W_\alpha \boldsymbol{a}(W,f)_{s(\alpha)}(x),$$

which is the main argument in the proof of the previous theorem, so the result applies to max-pooling. Note also that max-pooling vertices are positive scale invariant.

4.1. Consequences

Representing a neural network over a network quiver Q by a pair (W,f) and Theorem 1 has two consequences on neural networks.

4.1.1. Consequence 1

Corollary 1. *There are infinitely many neural networks with the same network function, independently of the architecture and the activation functions.*

If each neuron of a neural network is assigned a change of basis value $\tau_v \in \mathbb{C}$, its weights W can be transformed to another set of weights V following the group action of Equation (1). Similarly, the activation functions f of that network can be transformed to other ones g following the group action of Equation (2). For example, if f is ReLU and τ_v is a negative real value, then g becomes an inverted-flipped ReLU function, i.e., $min(0,x)$. From the usual neural network representation stand point, the two neural networks (W,f) and (V,g) are different as their activation functions f and g are different and their weights W and V are different. Nonetheless, their function (i.e., the output of the networks given some input vector x) is rigorously identical. This is true regardless of the structure of the neural network, its activation functions and weight vector W.

Said otherwise, Theorem 1 implies that there is not a unique neural network with a given network function and that an [infinite] amount of other neural networks with different weights and different activation functions have the same network function and that these other neural networks may be obtained with the change of basis group $\widetilde{G}$.

4.1.2. Consequence 2

A weak version of Theorem 1 proves a property of ReLU networks known as positive scale invariance or positive homogeneity [12,13,36–38]. Positive scale invariance is a property of ReLU non-linearities, where the network function remains unchanged if we (for example) multiply the weights in one layer of a network by a positive factor, and divide

the weights on the next layer by that same positive factor. Even more, this can be done on a per neuron basis. Namely, assigning a positive factor $r > 0$ to a neuron and multiplying every weight that points to that neuron with r, and dividing every weight that starts on that neuron by r.

Corollary 2 (Positive Scale Invariance of ReLU Networks)**.** *Let (W, f) be a neural network over Q over the real numbers where f is the ReLU activation function. Let $\tau = (\tau_v)_{v \in \mathcal{V}}$ where $\tau_v = 1$ if v is not a hidden vertex, and $\tau_v > 0$ for any other v. Then*

$$\tau \cdot (W, f) = (\tau \cdot W, f).$$

As a consequence, $(\tau \cdot W, f)$ and (W, f) are isomorphic neural networks. In particular, they have the same network function, $\Psi(\tau \cdot W, f) = \Psi(W, f)$.

Proof. Recall that $\tau \cdot (W, f) = (\tau \cdot W, \tau \cdot f)$. Since ReLU satisfies $f(\tau_v x) = \tau_v f(x)$ for all x and all $\tau_v > 0$ and since $(\tau \cdot f)$ corresponds to $\tau_v f\left(\frac{x}{\tau_v}\right)$ at each vertex v as mentioned in Equation (2), we get that $\tau_v f\left(\frac{x}{\tau_v}\right) = \frac{\tau_v}{\tau_v} f(x) = f(x)$ for each vertex v and thus $\tau \cdot f = f$. Finally, $\tau \cdot (W, f) = (\tau \cdot W, \tau \cdot f) = (\tau \cdot W, f)$. □

We stress that this known result is a consequence of neural networks being pairs (W, f) whose structure is governed by representation theory, and therefore exposes the algebraic and combinatorial nature of neural networks.

5. Architecture

In this section, we first outline the different types of architectures that we consider. We also show how the commonly used layers for neural networks translate into quiver representations. Finally, we will present in detail how an isomorphism of neural networks can be chosen so that the structure of the weights gets preserved.

5.1. Types of Architectures

Definition 20 ([3] (p. 193))**.** *The **architecture** of a neural network refers to its structure which accounts for how many units (neurons) it has and how these units are connected together.*

For our purposes, we distinguish three types of architectures: **combinatorial architecture**, **weight architecture** and **activation architecture**.

Definition 21. *The **combinatorial architecture** of a neural network is its network quiver. The **weight architecture** is given by constraints on how the weights are chosen, and the **activation architecture** is the set of activation functions assigned to the loops of the network quiver.*

If we consider the neural network of Figure 3, the combinatorial architecture specifies how the vertices are connected together, the weight architecture on how the weights W_ϵ are assigned and the activation architecture deals with the activation functions f_v.

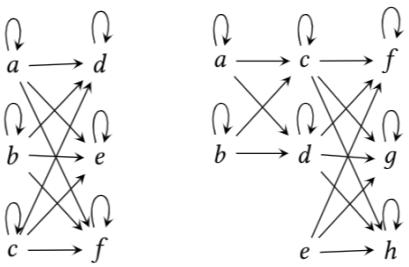
Two neural networks may have different combinatorial, weight and activation architecture like ResNet [7] vs. VGGnet [39] for example. Neural network layers may have the same combinatorial architecture but a different activation and weight architecture. It is the case for example of a mean pooling layer vs. a convolution layer. While they both encode a convolution (same combinatorial architecture) they have a different activation architecture (as opposed to conv layers, mean pooling has no activation function) and a different weight architecture as the mean pooling weights are fixed, and on conv layers they are shared across filters. This is what we mean by "*constraints*" on how the weights are chosen, namely, weights in conv layers and mean-pooling layers are not chosen freely, as in fully connected layers. Overall, two neural networks have globally the same architecture if and only if they share the same combinatorial, weight and activation architectures.

Additionally, isomorphic neural networks always have the same combinatorial architecture, since isomorphisms of neural networks are defined over the same network quiver. However, an isomorphism of neural networks can change or not the weight and the activation architecture. We will return to that concept at the end of this section.

5.2. Neural Network Layers

Here, we look at how fully-connected layers, convolutional layers, pooling layers, batch normalization layers and residual connections are related to the quiver representation language.

Let $\mathcal{V}^j$ be the set of vertices on the j-th hidden layer of Q. A **fully connected layer** is a hidden layer $\mathcal{V}^j$ where all vertices on the previous layer are connected to all vertices in $\mathcal{V}^j$. A **fully connected layer with bias** is a hidden layer $\mathcal{V}^j$ that puts constraints on the previous layer $\mathcal{V}^{j-1}$ such that the non-bias vertices of $\mathcal{V}^{j-1}$ are fully connected with the non-bias vertices of layer $\mathcal{V}^j$. A fully connected layer has no constraints on its weight and activation architecture but impose that the bias vertex has no activation function and not connected with the vertex of the previous layer. The reader can find an illustration of this in Figure 4.



(**a**) (**b**)

Figure 4. (**a**) Combinatorial architecture of a fully connected layer, one layer formed by a, b, c and the other by d, e, f. This architecture has no restrictions on the weight nor the activation architectures. (**b**) Two consecutive fully connected layers, first layer connecting a, b with c, d, and the second connecting c, d with f, g, h and e is a bias vertex. The first without bias and the second with bias. Note that there is no loop (activation function) on the bias vertex e.

A **convolutional layer** is a hidden layer $\mathcal{V}^j$ whose vertices are separated in channels (or feature maps). The weights are typically organized in filters $(F_n)_{n=1}^m$, and each F_n is a tensor made of channels. By "channels", we mean that the shape of, for example, a 2D convolution is given by $w \times h \times c$, where w is the width, h is the height and c is the number of channels on the previous layer. A "filter" is given by the weights and edges on a conv layer whose target lies in the same channel.

As opposed to fully connected layers, convolutional layers have constraints. One of which is that convolutional layers should be partitioned into channels of the same cardinality. Each filter F_n produces a channel on the layer $\mathcal{V}^j$ by a convolution of $\mathcal{V}^{j-1}$ with the filter F_n. Moreover, a convolution operation has a stride and may use padding.

A convolutional layer also has constraints on its combinatorial and weight architecture. First, each $\mathcal{V}^j$ is connected to a sub-set of vertices in the previous layer "in front" of which it is located. The combinatorial architecture of a conv layer for one feature map is illustrated in Figure 5a. Second, the weight architecture requires that the weights on the filters repeat in every sliding of the convolutional window. In other words, the weights of the edges on a conv layer must be shared across all filters as in Figure 5b.

A **conv layer with bias** is a hidden layer $\mathcal{V}^j$ partitioned into channels, where each channel is obtained by convolution of $\mathcal{V}^{j-1}$ with each filter F_n, $n = 1, \ldots, m$, plus one bias vertex in layer $\mathcal{V}^{j-1}$ that is connected to every vertex on every channel of $\mathcal{V}^j$. The weights of the edges starting on the bias vertex should repeat within the same channel. Again,

bias vertices do not have an activation function and are not connected to neurons of the previous layer.

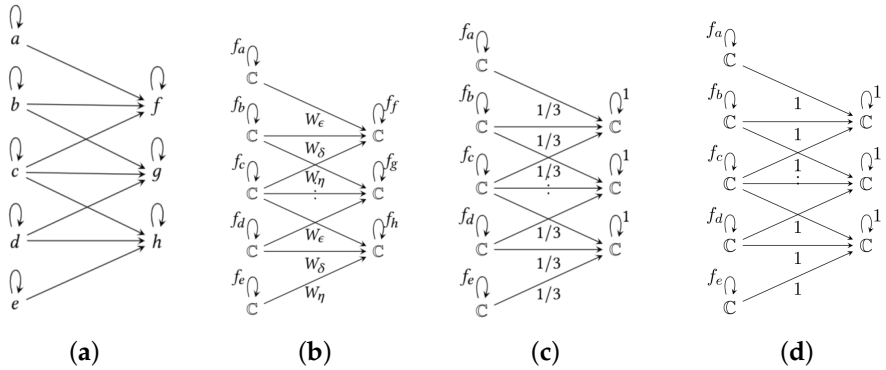


Figure 5. (**a**) Combinatorial architecture of a convolutional and a pooling layer. (**b**) Weight and activation architecture of a convolutional layer. (**c**) Weight and activation architecture of an average pooling layer. (**d**) Weight and activation architecture of a max-pooling layer.

The combinatorial architecture of a **pooling layer** is the same as that of a conv layer, see Figure 5a. However, since the purpose of that operation is usually to reduce the size of the previous layer, it contains non-trainable parameters. Thus, pooling layers have a different weight architecture than the conv layers. Average pooling fixes the weights in a layer to $1/n$ where n is the size of the feature map, while max-pooling fixes the weights in a layer to 1 and outputs the maximum over each window in the previous layer. Additionally, the activation function of an average and max-pooling layer is the identity function. This can be appreciated in Figure 5c,d.

Remark 4. *Max-pooling layers are compatible with our constructions, but they force us to consider another operation in the neuron, as was noted in Definition 15.*

It is known that max-pooling layers give a small amount of translation invariance at each level since the precise location of the most active feature detector is discarded, and this produces doubts about the use of max-pooling layers, see [40,41]. An alternative to this is the use of attention-based pooling [42], which is a global-average pooling. Our interpretation provides a framework that supports why these doubts about the use of max-pooling layers exist: they break the algebraic structure on the computations of a neural network. However, average pooling layers, and therefore global-average pooling layers, are perfectly consistent with respect to our results since they are given by fixed weights for any input vector while not requiring specification of another operation.

Batch normalization layers [43] require specifications on the three types of architecture. Their combinatorial architecture is given by two identical consecutive hidden layers where each neuron on the first is connected to only one neuron on the second, and there is one bias vertex in each layer. The weight architecture is given by the batch norm operation, which is $x \mapsto \dfrac{x-\mu}{\sigma^2}\gamma + \beta$ where μ is the mean of a batch and σ^2 its variance, and γ and β are learnable parameters. The activation architecture is given by two identity activations. This can be seen in Figure 6.

Remark 5. *The weights μ and σ are not determined until the network is fed with a batch of data. However, at test time, μ and σ are set to the overall mean and variance computed across the training data set and thus become normal weights. This does not mean that the architecture of the network depends on the input vector, but that the way these particular weights are chosen is by obtaining mean and variance from the data.*

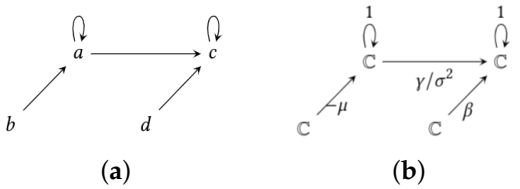


Figure 6. (**a**) Combinatorial architecture of a batch normalization layer. (**b**) Weight and activation architecture of a batch normalization layer. Observe that vertices b and d are bias vertices, and therefore the layer computes $x \mapsto x - \mu \mapsto (x - \mu)(\gamma/\sigma^2) + \beta$, which is the definition of the batch norm operation.

The combinatorial architecture of a **residual connection** [7] requires the existence of edges in Q that jump over one or more layers. Their weight architecture forces the weights chosen for those edges to be always equal to 1. We refer to Figure 7 for an illustration of the architecture of a residual connection.

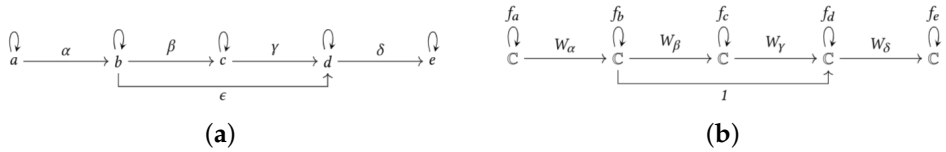
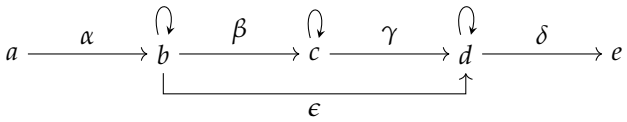


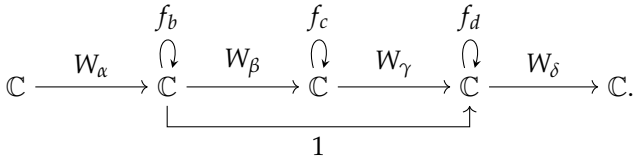
Figure 7. (**a**) Combinatorial architecture of a residual connection. (**b**) Weight architecture of a residual connection.

5.3. Architecture Preserved by Isomorphisms

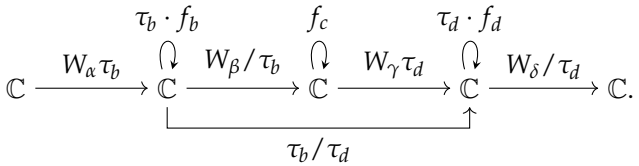
Two isomorphic neural networks can have different weight architectures. Let us illustrate this with a residual connection. Let Q be the following network quiver



and the neural network (W, f) over Q given by



Let $\tau_b \neq \tau_d$ be non-zero numbers, we define a change of basis of the neural network (W, f) by $\tau = (1, \tau_b, 1, \tau_d, 1)$. After applying the action of the change of basis $\tau \cdot (W, f)$ we obtain an isomorphic neural network given by



The neural networks (W, f) and $\tau \cdot (W, f)$ are isomorphic and therefore they have the same network function by Theorem 1. However, the neural network (W, f) has a residual connection, while $\tau \cdot (W, f)$ does not since the weight on the skip connection is not equal to 1. Nevertheless, if we take $\tau_b = \tau_d$, then the change of basis $\tau' = (1, \tau_b, 1, \tau_b, 1)$ will

produce an isomorphic neural network with a residual connection, and therefore both neural networks (W, f) and $\tau' \cdot (W, f)$ will have the same weight architecture.

The same phenomenon as for residual connections happens for convolutions, where one has to choose a specific kind of isomorphism to preserve the weight architecture, as shown in Figure 8. Isomorphisms of neural networks preserve the combinatorial architecture but not necessarily the weight architecture nor the activation architecture.

Remark 6. *Note that with the constructions given above any architecture can be written down as a neural network over a network quiver in the sense of Definition 14, such as multilayer perceptrons, VGG net, ResNet, DensNet, and so on.*

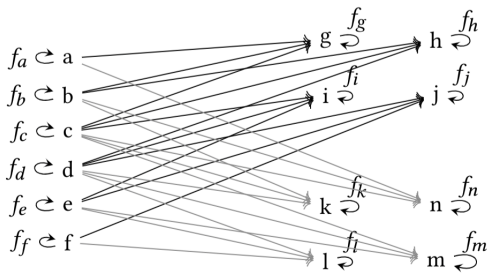


Figure 8. An illustration of a convolutional layer. The black arrows with target g, h, i and j correspond to the first channel, and the gray arrows with target k, l, m and n correspond to the second channel. A change of basis $\tau \in \widetilde{G}$ that preserves the weight architecture of this convolutional layer, has to be of the form $\tau = (\tau_i)_{i=a}^{m}$ where $\tau_g = \tau_h = \tau_i = \tau_j$ and $\tau_k = \tau_l = \tau_m = \tau_n$. Note that a convolution is given by filters which share weights, so if the previous condition is not satisfied, after applying the change of basis one will obtain weights that are not shared, so the resulting weights will not fit the definition of a convolution.

5.4. Consequences

As for the previous section, expressing neural network layers through the basic definitions of quiver representation theory has some consequences. Let us mention two.

5.4.1. Consequence 1

The first consequence derives from the isomorphism of residual layers. It is claimed by Meng et al. [13] that there is no positive scale invariance across residual blocks. However, we can see that the quiver representation language allows us to prove that in fact there is positive scale invariance across residual blocks for ReLU networks. Therefore, isomorphisms allow to understand that there are far more symmetries on neural networks than was previously known, as noted in Section 5.3, which can be written as follows:

Corollary 3. *There is invariance across residual blocks under isomorphisms of neural networks.*

5.4.2. Consequence 2

The second consequence is related to the existence of isomorphisms that preserve the weight architecture and not the activation architecture. As in Figure 8, a change of basis $\tau \in \widetilde{G}$, which preserves the weight architecture of this convolutional layer, has to be of the form $\tau = (\tau_i)_{i=a}^{m}$ where $\tau_g = \tau_h = \tau_i = \tau_j$ and $\tau_k = \tau_l = \tau_m = \tau_n$. This is what Meng et al. [13] do for the particular case of ReLU networks and positive change of basis (they consider the action of the group $\prod_{v \in \widetilde{Q}} \mathbb{R}_{>0}$ on neural networks). Note that if the change of basis is not chosen in this way, the isomorphism will produce a layer with different weights in each convolutional filter, and therefore the resulting operation will not be a convolution with respect to the same filter. While positive scale invariance of ReLU networks is a special kind of invariance under isomorphisms of neural networks that preserve both the weight and the activation architecture, we may generalize this

notion by allowing isomorphisms to change the activation architecture while preserving the weight architecture.

Definition 22. *Let (W, f) be a neural network and let $\tau \in \widetilde{G}$ be an element of the group of change of basis of neural networks such that the isomorphic neural network $\tau \cdot (W, f)$ has the same weight architecture as (W, f). The* ***teleportation*** *of the neural network (W, f) with respect to τ is the neural network $\tau \cdot (W, f)$.*

Since teleportation preserves the weight architecture, it follows that the teleportation of a conv layer is a conv layer, the teleportation of a pooling layer is a pooling layer, the teleportation of a batch norm layer is a batch norm layer, and the teleportation of a residual block is a residual block. Teleportation produces a neural network with the same combinatorial architecture, weight architecture and network function while it may change the activation architecture. For example, consider a neural network with ReLU activations and real change of basis. Since ReLU is positive scale invariant, any positive change of basis will leave ReLU invariant. On the other hand, for a negative change of basis the activation function changes to $min(0, x)$ and therefore the weight optimization landscape also changes. This implies that teleportation may change the optimization problem by changing the activation functions, while preserving the network function, and the network gets "*teleported*" to either other place in the same loss landscape (if the activation functions are not changed) or to a completely different loss landscape (if activation functions are changed).

6. Data Representations

In machine learning, a data sample is usually represented by a vector, a matrix or a tensor containing a series of observed variables. However, one may view data from a different perspective, namely the neuron outputs obtained after a forward pass, also known as "feature maps" for conv nets [3]. This has been done in the past to visualize what neurons have learned [11,21,23].

In this section, we propose a mathematical description of the data in terms of the architecture of the neural network, i.e., the neuron values obtained after a forward pass. We shall prove that doing so allows to represent data by a quiver representation. Our approach is different from representation learning [3] (p. 4) because we do not focus on how the representations are learned but rather on how the representations of the data are encoded by the forward pass of the neural network.

Definition 23. *A* ***labeled data set*** *is given by a finite set $D = \{(x_i, t_i)\}_{i=1}^n$ of pairs such that $x_i \in \mathbb{C}^d$ is a data vector (could also be a matrix or a tensor) and t_i is a target. We can have $t_i \in \mathbb{C}^k$ for a regression and $t_i \in \{C_0, C_1, \ldots, C_k\}$ for a classification.*

Let (W, f) be a neural network over a network quiver Q and a sample (x, t) of a data set D. When the network processes the input x, the vector x percolates through the edges and the vertices from the input to the output of the network. As mentioned before, this results in neuron values (or feature maps) that one can visualize [21]. On its own, the neuron values are not a quiver representation per se. However, one can combine these neuron values with their pre-activations and the network weights to obtain **a thin quiver representation**. Since that representation derives from the forward pass of x, it is specific to it. We will evaluate the activation functions in each neuron and then construct with them a quiver representation for a given input. We stress that this process is not ignoring the very important non-linearity of the activation functions, so no information of the forward pass is lost in this interpretation.

Remark 7. *Every thin quiver representation V of the delooped quiver Q° defines a neural network over the network quiver Q with identity activations, that we denote $(V, 1)$. We do not claim that taking identity activation functions for a neural network will result in something good in usual deep*

learning practices. This is only a theoretical trick to manipulate the underlying algebraic objects we have constructed. As such, we will identify thin quiver representations V with neural networks with identity activation functions $(V, 1)$.

Our data representation for x is a thin representation that we call $\mathtt{W}_x^f$ with identity activations whose function when fed with an input vector of ones $1^d := (1, \ldots, 1) \in \mathbb{C}^d$ satisfies

$$\Psi(\mathtt{W}_x^f, 1)(1^d) = \Psi(W, f)(x), \tag{4}$$

where $\Psi(W, f)(x)$ is the score of the network (W, f) after a forward pass of x.

Recovering $\mathtt{W}_x^f$ given the forward pass of x through (W, f) is illustrated in Figure 9a,b. Let us keep track of the computations of the network in the thin quiver representation $\mathtt{W}_x^f$ and remember that at the end, we want the output of the neural network $(\mathtt{W}_x^f, 1)$ when fed with the input vector $1^d \in \mathbb{C}^d$, to be equal to $\Psi(W, f)(x)$.

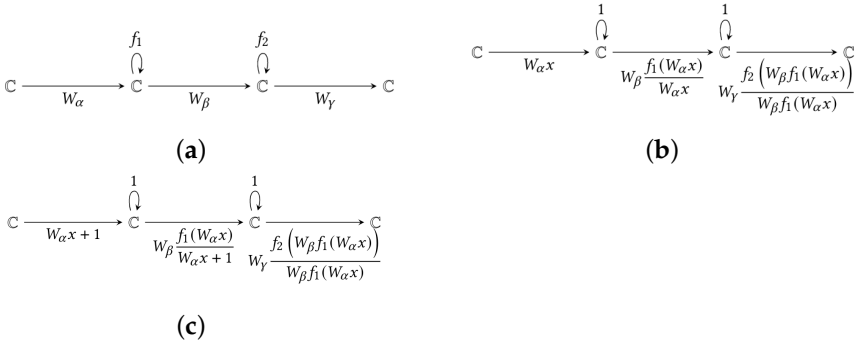


Figure 9. (**a**) A neural network (W,f). (**b**) The induced thin quiver representation $\mathtt{W}_x^f$ considered as a neural network $(\mathtt{W}_x^f, 1)$ and obtained after feed-forwarding x through (W, f). It can be seen that feed-forwarding a unit vector 1 through $\mathtt{W}_x^f$ (i.e., $\Psi(\mathtt{W}_x^f, 1)(1)$) gives the same output than feed-forwarding x through $(W, f) : \Psi(\mathtt{W}_x^f, 1)(1) = \Psi(W, f)(x)$. We refer to Theorem 2 for the general case. (**c**) In the case $W_\alpha x = 0$, we can add 1 to the corresponding pre-activation in $\mathtt{W}_x^f$ to prevent from a division by zero, while on the next layer we consider $W_\alpha x + 1$ as the pre-activation.

If $\epsilon \in \mathcal{E}$ is an oriented edge such that $s(\epsilon) \in \mathcal{V}$ is a bias vertex, then the computations of the weight corresponding to ϵ get encoded as $\left(\mathtt{W}_x^f\right)_\epsilon = W_\epsilon$. If, on the other hand, $s(\epsilon) \in \mathcal{V}$ is an input vertex, then the computations of the weights on the first layer get encoded as $\left(\mathtt{W}_x^f\right)_\epsilon = W_\epsilon x_{s(\epsilon)}$, see Figure 9b.

On the second and subsequent layers of the network (W, f) we encounter activation functions. Additionally, the weight corresponding to an oriented edge ϵ in $\mathtt{W}_x^f$ will have to cancel the unnecessary computations coming from the previous layer. That is, $\left(\mathtt{W}_x^f\right)_\epsilon$ has to be equal to W_ϵ times the activation output of the vertex $s(\epsilon)$ divided by the pre-activation of $s(\epsilon)$. Overall, $\mathtt{W}_x^f$ is defined as

$$\left(\mathtt{W}_x^f\right)_\epsilon = \begin{cases} W_\epsilon x_{s(\epsilon)} & \text{if } s(\epsilon) \text{ is an input vertex,} \\ W_\epsilon & \text{if } s(\epsilon) \text{ is a bias vertex,} \\ W_\epsilon \dfrac{\mathbf{a}(W, f)_{s(\epsilon)}(x)}{\sum_{\beta \in \zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W, f)_{s(\beta)}(x)} & \text{if } s(\epsilon) \text{ is a hidden vertex,} \end{cases} \tag{5}$$

where $\zeta_{s(\epsilon)}$ is the set of oriented edges of Q with target $s(\epsilon)$. In the case where the activation function is ReLU, for ϵ an oriented edge such that $s(\epsilon)$ is a hidden vertex, either $\left(\mathtt{W}_x^f\right)_\epsilon = 0$ or $\left(\mathtt{W}_x^f\right)_\epsilon = W_\epsilon$.

Remark 8. *Observe that the denominator* $\sum_{\beta\in\zeta_{s(\epsilon)}} W_\beta \cdot \boldsymbol{a}(W,f)_{s(\beta)}(x)$ *is the pre-activation of vertex* $s(\epsilon)$ *and can be equal to zero. However, the set where this happens is of measure zero. Even in the case that it turns out to be exactly zero, one can add a number* $\eta \neq 0$ *(for example* $\eta = 1$*) to make it non-zero and then consider* η *as the pre-activation of that corresponding neuron, see Figure 9c. Therefore, we will assume, without loss of generality, that pre-activations of neurons are always non-zero.*

The quiver representation $\mathtt{W}_x^f$ of the delooped quiver Q° accounts for the combinatorics of the history of all the computations that the neural network (W,f) performs on a forward pass given the input x. The main property of the quiver representation $\mathtt{W}_x^f$ is given by the following result. A small example of the computation of $\mathtt{W}_x^f$ and a view into how the next Theorem works can be found in Appendix B.

Theorem 2. *Let* (W,f) *be a neural network over* Q*, let* (x,t) *be a data sample for* (W,f) *and consider the induced thin quiver representation* $\mathtt{W}_x^f$ *of* Q°*. The network function of the neural network* $(\mathtt{W}_x^f, 1)$ *satisfies*

$$\Psi(\mathtt{W}_x^f, 1)(1^d) = \Psi(W,f)(x).$$

Proof. Obviously, both neural networks have different input vectors, that is, 1^d for $(\mathtt{W}_x^f,1)$ and x for (W,f). If $v \in \mathcal{V}$ is a source vertex, by definition $\mathbf{a}(\mathtt{W}_x^f,1)_v(1^d) = 1$. We will show that in the other layers, the activation output of a vertex in $(\mathtt{W}_x^f,1)$ is equal to the pre-activation of (W,f) in that same vertex. Assume that $v \in \mathcal{V}$ is in the first hidden layer, let ζ_v^{bias} be the set of oriented edges of Q with target v and source vertex a bias vertex, and let ζ_v^{input} be the set of oriented edges of Q with target v and source vertex an input vertex. Then, for every $\epsilon \in \zeta_v$ where $\zeta_v = \zeta_v^{bias} \cup \zeta_v^{input}$, we have that $\mathbf{a}(\mathtt{W}_x^f,1)_{s(\epsilon)}(1^d) = 1$, and therefore

$$\begin{aligned}
\mathbf{a}(\mathtt{W}_x^f,1)_v(1^d) &= \sum_{\epsilon\in\zeta_v} \left(\mathtt{W}_x^f\right)_\epsilon \mathbf{a}(\mathtt{W}_x^f,1)_{s(\epsilon)}(1^d) \\
&= \sum_{\epsilon\in\zeta_v} \left(\mathtt{W}_x^f\right)_\epsilon \\
&= \sum_{\epsilon\in\zeta_v^{bias}} \left(\mathtt{W}_x^f\right)_\epsilon + \sum_{\epsilon\in\zeta_v^{input}} \left(\mathtt{W}_x^f\right)_\epsilon \\
&= \sum_{\epsilon\in\zeta_v^{bias}} W_\epsilon + \sum_{\epsilon\in\zeta_v^{input}} W_\epsilon x_{s(\epsilon)},
\end{aligned}$$

which is the pre-activation of vertex v in (W,f), i.e., $f_v\left(\mathbf{a}(\mathtt{W}_x^f,1)_v(1^d)\right) = \mathbf{a}(W,f)_v(x)$. If $v \in \mathcal{V}$ is in the second hidden layer then

$$\begin{aligned}
\mathbf{a}(\mathtt{W}_x^f,1)_v(1^d) &= \sum_{\epsilon\in\zeta_v} \left(\mathtt{W}_x^f\right)_\epsilon \mathbf{a}(\mathtt{W}_x^f,1)_{s(\epsilon)}(1^d) \\
&= \sum_{\epsilon\in\zeta_v} W_\epsilon \frac{\mathbf{a}(W,f)_{s(\epsilon)}(x)}{\sum_{\beta\in\zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W,f)_{s(\beta)}(x)} \mathbf{a}(\mathtt{W}_x^f,1)_{s(\epsilon)}(1^d)
\end{aligned}$$

since $\sum_{\beta\in\zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W,f)_{s(\beta)}(x)$ is the pre-activation of vertex $s(\epsilon)$ in (W,f), by the above formula we get that $\sum_{\beta\in\zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W,f)_{s(\beta)}(x) = \mathbf{a}(\mathtt{W}_x^f,1)_{s(\epsilon)}(1^d)$, and then

$$\mathbf{a}(\mathtt{W}_x^f,1)_v(1^d) = \sum_{\epsilon\in\zeta_v} W_\epsilon \mathbf{a}(W,f)_{s(\epsilon)}(x),$$

which is the pre-activation of vertex v in (W, f) when fed with the input vector x. That is, $f_v\left(\mathbf{a}(\mathtt{W}_x^f, 1)_v(1^d)\right) = \mathbf{a}(W, f)_v(x)$. An induction argument gives the desired result since the output layer has no activation function, and the coordinates of $\Upsilon(\mathtt{W}_x^f, 1)(1^d)$ and $\Upsilon(W, f)(x)$ are the values of the output vertices. □

6.1. Consequences

Interpreting data as quiver representations has several consequences.

6.1.1. Consequence 1

The combinatorial architecture of (W, f) and of $(\mathtt{W}_x^f, 1)$ are equal, and the weight architecture of $(\mathtt{W}_x^f, 1)$ is determined by both the weight and activation architectures of the neural network (W, f) when it is fed the input vector x. This means that even though the network function is non-linear because of the activation functions, all computations of the forward pass of a network on a given input vector can be arranged into a linear object (the quiver representation $\mathtt{W}_x^f$), while preserving the output of the network, by Theorem 2.

Even more, feature maps and outputs of hidden neurons can be recovered completely from the quiver representations $\mathtt{W}_x^f$, which implies that the notion [11,23] of *representation created by a neural network* in deep learning is a mathematical consequence of understanding data as quiver representations.

It is well known that feature maps can be visualized into images showing the input signal characteristics and thus providing intuitions on the behavior of the network and its impact on an image [11,21–23]. This notion is implied by our findings as our thin quiver representations of data $\mathtt{W}_x^f$ include both the network structure and the feature maps induced by the data, expressed by the formula

$$f_v\left(\mathbf{a}(\mathtt{W}_x^f, 1)_v(1^d)\right) = \mathbf{a}(W, f)_v(x),$$

see Equation (5) in page 19 and the proof of Theorem 2.

Practically speaking, it is useless to compute the quiver representation $\mathtt{W}_x^f$ only to recover the outputs of hidden neurons, that are even more efficiently computed directly from the forward pass of data. Nevertheless, the way in which the outputs of hidden neurons are obtained from the quiver representations $\mathtt{W}_x^f$ is by forgetting algebraic structure, more specifically forgetting pieces of the quiver, which is formalized by the notion of *forgetful functors* in representation theory. All this implies that the notion of representation in deep learning is obtained from the quiver representations $\mathtt{W}_x^f$ by loosing information of the computations of the neural network.

As such, using a thin quiver representation opens the door to a formal (and less intuitive) way to understand the interaction between data and the structure of a network, that takes into account all the combinatorics of the network and not only the activation outputs of the neurons, as it is currently understood.

6.1.2. Consequence 2

Corollary 4. *Let (x, t) and (x', t') be data samples for (W, f). If the quiver representations $\mathtt{W}_x^f$ and $\mathtt{W}_{x'}^f$ are isomorphic via $\widetilde{G}$ then $\Upsilon(W, f)(x) = \Upsilon(W, f)(x')$.*

Proof. The neural networks $(\mathtt{W}_x^f, 1)$ and $(\mathtt{W}_{x'}^f, 1)$ are isomorphic if and only if the quiver representations $\mathtt{W}_x^f$ and $\mathtt{W}_{x'}^f$ are isomorphic via $\widetilde{G}$. By the last Theorem and the fact that isomorphic neural networks have the same network function (Theorem 1) we obtain

$$\Upsilon(W, f)(x) = \Upsilon(\mathtt{W}_x^f, 1)(1^d) = \Upsilon(\mathtt{W}_{x'}^f, 1)(1^d) = \Upsilon(W, f)(x').$$

□

By this Corollary and the invariance of the network function under isomorphisms of the group $\widetilde{G}$ (Theorem 1), we obtain that the neural network is representing the data and the output on (W, f) as the isomorphism classes $\left[\mathtt{W}_x^f\right] := \{\tau \cdot \mathtt{W}_x^f : \tau \in \widetilde{G}\}$ of the thin quiver representations $\mathtt{W}_x^f$ under the action of the change of basis group $\widetilde{G}$ of neural networks. This motivates the construction of a space whose points are isomorphism classes of quiver representations, which is exactly the construction of "*moduli space*" presented in the next section.

6.2. Induced Inquiry for Future Research

The language of quiver representations applied to neural networks brings new perspectives on their behavior and thus is likely to open doors for future works. Here is one inquiry for the future.

If a data sample x is represented by a thin quiver representation $\mathtt{W}_x^f$, one can generate an infinite amount of new data representations $\mathtt{W}_{x'}^f$ via $\widetilde{G}$ which all have the same network output, by applying an isomorphism given by $\tau \in \widetilde{G}$ using Equation (1), and then constructing an input x' from it that produces such isomorphic quiver representation. Doing so could have important implications in the field of adversarial attacks and network fooling [44] where one could generate fake data at will which, when fed to a network, all have exactly the same output as the original data x. This will require the construction of a map from quiver representations to the input space, which could be done by using tools from algebraic geometry to find sections of the map $x \mapsto \left[\mathtt{W}_x^f\right]$, for which the construction of the moduli space in the next section is necessary, but not sufficient. This leads us to propose the following question for future research:

"Can the data quiver representations $\mathtt{W}_x^f$ be translated back to input data?"

Following the same logic, one could use this for data augmentation. Starting from an annotated dataset $D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$, one could represent each data x_i by a thin quiver representation : $\mathtt{W}_{x_i}^f$, apply an arbitrary number of isomorphisms to it: $\{\tau^1 \mathtt{W}_{x_i}^f, \tau^2 \mathtt{W}_{x_i}^f, \ldots, \tau^M \mathtt{W}_{x_i}^f\}$ and then convert these representations back to the input data space.

7. The Moduli Space of a Neural Network

In this section, we propose a modified version of the manifold hypothesis of Goodfellow et al. [3] (Section 5.11.3). The original manifold hypothesis claims that the data lie in a small dimensional manifold inside the input space. We will provide an explicit map from the input space to the moduli space of a neural network with which the data manifold can be translated to the moduli space. This will allow the use of mathematical theory for quiver moduli spaces [25–27] to manifold learning, representation learning and the dynamics of neural network learning [11,23].

Remark 9. *Throughout this section, we assume that all the weights of a neural network and of the induced data representations $\mathtt{W}_x^f$ are non-zero. This can be assumed since the set where some of the weights are zero is of measure zero, and even in the case where it is exactly zero we can add a small number to it to make it non-zero and at the same time imperceptible to the computations of any computer, for example, infinitesimally smaller than the machine epsilon.*

In order to formalize our manifold hypothesis, we will attach an explicit geometrical object to every neural network (W, f) over a network quiver Q, that will contain the isomorphism classes of the data quiver representations $\left[\mathtt{W}_x^f\right]$ induced by any kind of data set D. This geometrical object that we denote ${}_d\mathcal{M}_k(\tilde{Q})$ is called the **moduli space**. The moduli space only depends on the combinatorial architecture of the neural network,

while the activation and weight architectures of the neural network determine how the isomorphism classes of the data quiver representations $\left[\mathtt{W}_x^f\right]$ are distributed inside the moduli space.

The mathematical objects required to formalize our manifold hypothesis are known as **framed quiver representations**. We will follow Reineke [25] for the construction of framed quiver representations in our particular case of thin representations. Recall that the **hidden quiver** $\widetilde{Q} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{s}, \widetilde{t})$ of a network quiver Q is the sub-quiver of the delooped quiver Q° formed by the hidden vertices $\widetilde{\mathcal{V}}$ and the oriented edges $\widetilde{\mathcal{E}}$ between hidden vertices. Every thin representation of the delooped quiver Q° induces a thin representation of the hidden quiver $\widetilde{Q}$ by forgetting the oriented edges whose source is an input (or bias) vertex, or the target is an output vertex.

Definition 24. *We call **input vertices** of $\widetilde{Q}$ the vertices of $\widetilde{Q}$ that are connected to the input vertices of Q, and we call **output vertices** of $\widetilde{Q}$ the vertices that are connected to the output vertices of Q.*

Observe that the input vertices of the hidden quiver $\widetilde{Q}$ may not all of them be source vertices, so in the neural network we allow oriented edges from the input layer to deeper layers in the network. Dually, the output vertices of the hidden quiver $\widetilde{Q}$ may not all of them be sink vertices, so in the neural network we allow oriented edges from any layer to the output layer.

Remark 10. *For the sake of simplicity, we will assume that there are no bias vertices in the quiver Q. If there are bias vertices in Q, we can consider them as part of the input layer in such a way that every input vector $x \in \mathbb{C}^d$ needs to be extended to a vector $x' \in \mathbb{C}^{d+b}$ with its last b coordinates all equal to 1, where b is the number of bias vertices. All the quiver representation theoretic arguments made in this section are therefore valid also for neural networks with bias vertices under these considerations. This also has to do with the fact that the group of change of basis of neural networks $\widetilde{G}$ has no factor corresponding to bias vertices, as the hidden quiver is obtained by removing all source vertices, not only input vertices.*

Let $\widetilde{W}$ be a thin representation of $\widetilde{Q}$. We fix once and for all a family of vector spaces $\{V_v\}_{v\in\widetilde{\mathcal{V}}}$ indexed by the vertices of $\widetilde{Q}$, given by $V_v = \mathbb{C}^k$ when v is an output vertex of $\widetilde{Q}$ and $V_v = 0$ for any other $v \in \widetilde{\mathcal{V}}$.

Definition 25 ([25])**.** *A choice of a thin representation $\widetilde{W}$ of the hidden quiver and a map $h_v : \widetilde{W}_v \to V_v$ for each $v \in \widetilde{\mathcal{V}}$ determines a pair $(\widetilde{W}, h)$, where $h = \{h_v\}_{v\in\widetilde{\mathcal{V}}}$, that is known as a **framed** quiver representation of $\widetilde{Q}$ by the family of vector spaces $\{V_v\}_{v\in\widetilde{\mathcal{V}}}$.*

We can see that h_v is equal to the zero map when v is not an output vertex of $\widetilde{Q}$, and $h_v : \mathbb{C} \to \mathbb{C}^k$ for every v output vertex of $\widetilde{Q}$.

Dually, we can fix a family of vector spaces $\{U_v\}_{v\in\widetilde{\mathcal{V}}}$ indexed by $\widetilde{\mathcal{V}}$ and given by $U_v = \mathbb{C}^d$ when v is an input vertex of $\widetilde{Q}$ and $U_v = 0$ for any other $v \in \widetilde{\mathcal{V}}$.

Definition 26 ([25])**.** *A choice of a thin representation $\widetilde{W}$ of the hidden quiver and a map $\ell_v : U_v \to \widetilde{W}_v$ for each $v \in \widetilde{\mathcal{V}}$ determines a pair $(\widetilde{W}, \ell)$, where $\ell = \{\ell_v\}_{v\in\widetilde{\mathcal{V}}}$, that is known as a **co-framed** quiver representation of $\widetilde{Q}$ by the family of vector spaces $\{U_v\}_{v\in\widetilde{\mathcal{V}}}$.*

We can see that ℓ_v is the zero map when v is not an input vertex of $\widetilde{Q}$, and $\ell_v : \mathbb{C}^d \to \mathbb{C}$ for every v an input vertex of $\widetilde{Q}$.

Definition 27. *A **double-framed** thin quiver representation is a triple $(\ell, \widetilde{W}, h)$ where $\widetilde{W}$ is a thin quiver representation of the hidden quiver, $(\widetilde{W}, h)$ is a framed representation of $\widetilde{Q}$ and $(\widetilde{W}, \ell)$ is a co-framed representation of $\widetilde{Q}$.*

Remark 11. *In representation theory, one does either a framing or a co-framing, and chooses a stability condition for each one. In our case, we will do both at the same time, and use the definition of stability given by [25] for framed representations, together with its dual notion of stability for co-framed representations.*

Definition 28. *The group of **change of basis of double-framed thin quiver representations** is the same group $\widetilde{G}$ of change of basis of neural networks.*

The action of $\widetilde{G}$ on double-framed quiver representations for $\tau \in \widetilde{G}$ is given by

$$\tau \cdot (\ell, \widetilde{W}, h) = (\tau \cdot \ell, \tau \cdot \widetilde{W}, \tau \cdot h),$$

where each component of $\tau \cdot h$ is given by $(\tau \cdot h)_v := (h_v^1/\tau_v, \ldots, h_v^k/\tau_v)$, if we express $h_v = (h_v^1, \ldots, h_v^k)$, and each component of $\tau \cdot \ell$ is given by $(\tau \cdot \ell)_v := (\ell_v^1\tau_v, \ldots, \ell_v^k\tau_v)$, if we express $\ell_v = (\ell_v^1, \ldots, \ell_v^k)$. Every double-framed thin quiver representation of $\widetilde{Q}$ isomorphic to $(\ell, \widetilde{W}, h)$ is of the form $\tau \cdot (\ell, \widetilde{W}, h)$ for some $\tau \in \widetilde{G}$. In the following theorem, we show that instead of studying the isomorphism classes $\left[\mathtt{W}_x^f\right]$ of the thin quiver representations of the delooped quiver Q° induced by the data, we can study the isomorphism classes of double-framed thin quiver representations of the hidden quiver.

Theorem 3. *There exists a bijective correspondence between the set of isomorphism classes $[W]$ via $\widetilde{G}$ of thin representations over the delooped quiver Q° and the set of isomorphism classes $[(\ell, \widetilde{W}, h)]$ of double-framed thin quiver representations of $\widetilde{Q}$.*

Proof. The correspondence between isomorphism classes is due to the equality of the group of change of basis for neural networks and double-framed thin quiver representations, since the isomorphism classes are given by the action of the same group. Given a thin representation W of the delooped quiver, it induces a thin representation $\widetilde{W}$ of the hidden quiver $\widetilde{Q}$ by forgetting the input and output layers of Q. Moreover, if we consider the input vertices of Q as the coordinates of $\mathbb{C}^d$ and the output vertices of Q as the coordinates of $\mathbb{C}^k$, then the weights starting on input vertices of Q define the map ℓ while the weights ending on output vertices of Q define the map h. This can be seen in Figure 10. Given a double-framed thin quiver representation $(\ell, \widetilde{W}, h)$, the entries ℓ (resp., h) are the weights of a thin representation W starting (resp., ending) on input (resp., output) vertices, while $\widetilde{W}$ defines the hidden weights of W. □

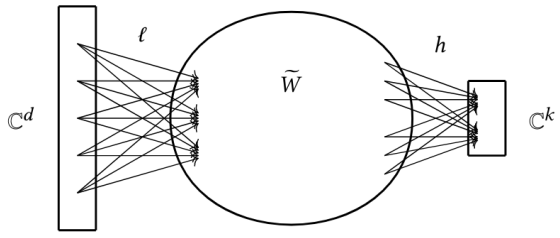


Figure 10. An illustration of a double-framed thin quiver representation $(\ell, \widetilde{W}, h)$. The boxes define the vector spaces of the framing and co-framing, given by $\mathbb{C}^d$ and $\mathbb{C}^k$, respectively.

From now on, we will identify a double-framed thin quiver representation $(\ell, \widetilde{W}, h)$ with the thin representation W of the delooped quiver Q° defined by $(\ell, \widetilde{W}, h)$ as in the proof of the last theorem. We will also identify the isomorphism classes

$$[W] = \Big[(\ell, \widetilde{W}, h)\Big],$$

where the symbol on the left means the isomorphism class of the thin representation W under the action of $\widetilde{G}$, and the one on the right is the isomorphism class of the double-framed thin quiver representation $(\ell, \widetilde{W}, h)$.

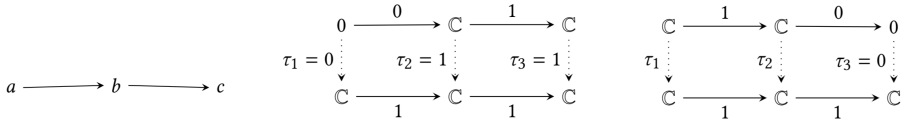
One would like to study the space of *all* isomorphism classes of double-framed thin representations of the delooped quiver. However, it is well known that this space does not have a good topology [45]. Therefore, one considers the space of isomorphism classes of *stable* double-framed thin quiver representations instead of all quiver representations, which can be shown to have a much richer topological and geometrical structure. In order to be stable, a representation has to satisfy a stability condition that is given in terms of its *sub-representations*. We will prove that the data representations $\mathtt{W}_x^f$ are stable in this sense, and to do so we will now introduce the necessary definitions.

Definition 29 ([10] (p. 14))**.** *Let W be a thin representation of the delooped quiver Q° of a network quiver Q. A **sub-representation** of W is a representation U of Q° such that there is a morphism of representations $\tau : U \to W$ where each map τ_v is an injective map.*

Definition 30. *The **zero representation** of Q is the representation denoted 0 where every vector space assigned to every vertex is the zero vector space, and therefore every linear map in it is also zero.*

Note that if U is a quiver representation, then the zero representation 0 is a sub-representation of U since $\tau_v = 0$ is an injective map in this case.

We can see from Figure 11 that the combinatorics of the quiver are related to the existence of sub-representations. Therefore, we explain now how to use the combinatorics of the quiver to prove stability of our data-representations $\mathtt{W}_x^f$.



(**a**) (**b**) (**c**)

Figure 11. (**a**) A quiver Q. (**b**) A diagram showing a quiver representation of Q at the bottom and a sub-representation at the top together with the morphism of representations τ from Definition 29 given by dotted oriented edges. (**c**) A diagram showing a quiver representation of Q at the bottom and another one at the top. The representation at the top is not a sub-representation of the bottom representation because there is no possible choice of morphism of representations τ that is injective in every vertex.

Given a double-framed thin quiver representation $(\ell, \widetilde{W}, h)$, the image of the map ℓ lies inside the representation $\widetilde{W}$. The map ℓ is given by a family of maps indexed by the vertices of the hidden quiver $\widetilde{Q}$, namely, $\ell = \{\ell_v : \mathbb{C}^{n_v} \to \widetilde{W}_v \mid v \in \widetilde{\mathcal{V}}\}$. Recall that $n_v = 0$ if v is not an input vertex of the hidden quiver $\widetilde{Q}$, and $n_v = d$ when v is an input vertex of $\widetilde{Q}$. The image of ℓ is by definition a family of vector spaces indexed by the hidden quiver $\widetilde{Q}$, given by

$$Im(\ell) = \big(Im(\ell)_v\big)_{v \in \widetilde{\mathcal{V}}} \text{ where } Im(\ell)_v \subset \widetilde{W}_v.$$

By definition, $Im(\ell)_v = \{z \in \widetilde{W}_v = \mathbb{C} \mid \ell_v(w) = z \text{ for some } w \in \mathbb{C}^{n_v}\}$. Recall that we will interpret the data quiver representations $\mathtt{W}_x^f$ as double-framed representations, and that

$\mathtt{W}_x^f$ respects the output of the network (W, f) when it is fed the input vector x. According to Equation (5), the weights in the input layer of $\mathtt{W}_x^f$ are given in terms of the weights in the input layer of the network W and the input vector x. Therefore, only on a set of measure zero we have that some of the weights in the input layer of $\mathtt{W}_x^f$ are zero, so we can assume, without loss of generality, that the weights on the input layer of $\mathtt{W}_x^f$ are all non-zero.

Dually, the kernel of the map h lies inside the representation $\widetilde{W}$. The map h is given by a family of maps indexed by the vertices of the hidden quiver $\widetilde{Q}$, namely $h = \{h_v : \widetilde{W}_v \to \mathbb{C}^{m_v} \mid v \in \widetilde{\mathcal{V}}\}$. Recall that $m_v = 0$ if v is not an output vertex of the hidden quiver $\widetilde{Q}$, and $m_v = k$ when v is an output vertex of $\widetilde{Q}$. Therefore, the kernel of h is by definition a family of vector spaces indexed by the hidden quiver $\widetilde{Q}$. That is,

$$ker(h) = \big(ker(h)_v\big)_{v\in\widetilde{\mathcal{V}}} \text{ where } ker(h)_v \subset \widetilde{W}_v.$$

By definition $ker(h)_v = \{z \in \widetilde{W}_v \mid h_v(z) = 0\}$. The set where all of h_v are equal to zero is of measure zero, and even in the case where it is exactly zero we can add a very small number to every coordinate of h_v to make it non-zero and that the output of the network does not change significantly. Thus, we can assume, without loss of generality, that all the maps h_v are non-zero for every output vertex v of $\widetilde{Q}$.

Definition 31. *A double-framed thin quiver representation $(\ell, \widetilde{W}, h)$ is **stable** if the following two conditions are satisfied:*

1. *The only sub-representation U of $\widetilde{W}$ which is contained in $ker(h)$ is the zero sub-representation, and*
2. *The only sub-representation U of $\widetilde{W}$ that contains $Im(\ell)$ is $\widetilde{W}$.*

Theorem 4. *Let (W, f) be a neural network and let (x, t) be a data sample for (W, f). Then the double-framed thin quiver representation $\mathtt{W}_x^f$ is stable.*

Proof. We express $\mathtt{W}_x^f = (\ell, \widetilde{W}, h)$ as in Theorem 3. As explained before Definition 31, we can assume, without loss of generality, that for every input vertex v of $\widetilde{Q}$ the map ℓ_v is non-zero, and that for every output vertex v of $\widetilde{Q}$ the map h_v is non-zero.

We have that $h_v : \mathbb{C} \to \mathbb{C}^k$ is a linear map, so its kernel is either 0 or $\mathbb{C}$. However, $Ker(h_v) = \mathbb{C}$ if and only if $h_v = 0$, and since $h_v \neq 0$ we get that $Ker(h_v) = 0$ and, as in Figure 11, after the combinatorics of quiver representations, there is no sub-representation of $\widetilde{W}$ with all its factors corresponding to output vertices of $\widetilde{Q}$, other than the zero representation. Since the combinatorics of network quivers forces a sub-representation contained in $ker(h)$ to be the zero sub-representation, we obtain the first condition for stability of double-framed thin quiver representations.

Dually, we have that $\ell_v : \mathbb{C}^d \to \mathbb{C}$ is a linear map, so its image is either 0 or $\mathbb{C}$. However, $Im(\ell_v) = 0$ if and only if $\ell_v = 0$, and since $\ell_v \neq 0$ we get that $Im(\ell_v) = \mathbb{C}$ and, as in Figure 11, there is no sub-representation of $\widetilde{W}$ that contains $Im(\ell)$ other than $\widetilde{W}$. Therefore, the only sub-representation of $\widetilde{W}$ that contains $Im(\ell)$ is $\widetilde{W}$.

Thus, $\mathtt{W}_x^f = (\ell, \widetilde{W}, h)$ is a stable double-framed thin quiver representation of the hidden quiver $\widetilde{Q}$. □

Denote by ${}_d\mathcal{R}_k(\widetilde{Q})$ the space of all double-framed thin quiver representations.

Definition 32. *The **moduli space** of stable double-framed thin quiver representations of $\widetilde{Q}$ is by definition*

$${}_d\mathcal{M}_k(\widetilde{Q}) := \big\{[V] : V \in {}_d\mathcal{R}_k(\widetilde{Q}) \text{ is stable}\big\}.$$

Note that the moduli space depends on the hidden quiver $\widetilde{Q}$ and the chosen vector spaces from which one double-frames the thin representations.

Given a neural network (W, f) and an input vector $x \in \mathbb{C}^d$, we can define a map

$$\begin{array}{rccc} \varphi(W,f): & \mathbb{C}^d & \to & {}_d\mathcal{R}_k(\widetilde{Q}) \\ & x & \mapsto & \mathtt{W}_x^f. \end{array}$$

By the last theorem, in the case where all the weights of $\mathtt{W}_x^f$ are non-zero, this map takes values in the moduli space which parametrizes isomorphism classes of stable double-framed thin quiver representations

$$\varphi(W,f): \quad \mathbb{C}^d \quad \to \quad {}_d\mathcal{M}_k(\widetilde{Q}).$$

Remark 12. *For ReLU activations one can produce representations with some weights* $\left(\mathtt{W}_x^f\right)_\epsilon = 0$. *However, note that these representations* $\mathtt{W}_x^f$ *can be arbitrarily approximated by representations with non-zero weights. Nevertheless, the map* $\varphi(W,f)$ *with values in* ${}_d\mathcal{R}_k(\widetilde{Q})$ *still decomposes the network function as in Consequence 1 below.*

The following result is a particular case of Nakajima [45]'stheorem, generalized for double-framings and restricted to thin representations, combined with Reineke [25]'s calculation of framed quiver moduli space dimension adjusted for double-framings (see Appendix C for details about the computation of this dimension).

Theorem 5. *Let* Q *be a network quiver. There exists a geometric quotient* ${}_d\mathcal{M}_k(\widetilde{Q})$ *by the action of the group* $\widetilde{G}$, *called the* ***moduli space*** *of stable double-framed thin quiver representations of* $\widetilde{Q}$. *Moreover,* ${}_d\mathcal{M}_k(\widetilde{Q})$ *is non-empty and its complex dimension is*

$$dim_{\mathbb{C}}\left({}_d\mathcal{M}_k(\widetilde{Q})\right) = \#\mathcal{E}^\circ - \#\widetilde{\mathcal{V}}.$$

In short, the dimension of the moduli space of the hidden quiver $\widetilde{Q}$ *equals the number of edges of* Q° *minus the number of hidden vertices.*

Remark 13. *The mathematical existence of the moduli space [25,45] depends on two things,*

- *the neural networks and the data may be build upon the real numbers, but we are considering them over the complex numbers, and*
- *the change of basis group of neural networks* $\widetilde{G}$ *is the change of basis group of thin quiver representations of* $\widetilde{Q}$, *which is a reductive group.*

One may try to study instead the space whose points are isomorphism classes given by the action of the sub-group H *of the change of basis group* $\widetilde{G}$, *whose action preserves both the weight and the activation architectures. By doing so we obtain a group* H *that is not reductive, which gets in the way of the construction, and therefore the existence, of the moduli space. This happens even in the case of ReLU activation.*

Finally, let us underline that the map $\varphi(W,f)$ from the input space to the representation space (i) takes values in the moduli space when all weights of the representations $\mathtt{W}_x^f$ are non-zero, and (ii) may or may not be 1-1. Even if $\varphi(W,f)$ is not 1-1, all the results in this work still hold. The most important implication of the existence of the map $\varphi(W,f)$ is our Consequence 1 below, which does not depend on $\varphi(W,f)$ being 1-1.

7.1. Consequences

The existence of the moduli space of a neural network has the following consequences.

7.1.1. Consequence 1

The moduli space ${}_d\mathcal{M}_k(\widetilde{Q})$ as a set is given by

$$ {}_d\mathcal{M}_k(\widetilde{Q}) = \Big\{ [V] : V \in {}_d\mathcal{R}_k(\widetilde{Q}) \text{ is stable} \Big\}. $$

That is, the points of the moduli space are the isomorphism classes of (stable) double-framed thin quiver representations of $\widetilde{Q}$ over the action of the change of basis group $\widetilde{G}$ of neural networks. Given any point in the moduli space $[V]$ we can define

$$ \hat{\Psi}[V] := \Psi(V,1)(1^d) $$

since the network function is invariant under isomorphisms, which gives a map

$$ \hat{\Psi} :_d \mathcal{M}_k(\widetilde{Q}) \to \mathbb{C}^k. $$

Furthermore, given a neural network (W,f), we define a map $\varphi(W,f) : \mathbb{C}^d \to_d \mathcal{M}_k(\widetilde{Q})$ by

$$ \varphi(W,f)(x) := \left[\mathtt{W}_x^f \right] \in {}_d\mathcal{M}_k(\widetilde{Q}). $$

Corollary 5. *The network function of any neural network (W,f) is decomposed as*

$$ \Psi(W,f) = \hat{\Psi} \circ \varphi(W,f). $$

Proof. This is a consequence of Theorem 2 since for any $x \in \mathbb{C}$ we have

$$ \hat{\Psi} \circ \varphi(W,f)(x) = \hat{\Psi}\left[\mathtt{W}_x^f \right] = \Psi(\mathtt{W}_x^f, 1)(1^d) = \Psi(W,f)(x). $$

□

This implies that any decision of any neural network passes through the moduli space (and the representation space), and this fact is independent of the architecture, the activation function, the data and the task.

7.1.2. Consequence 2

Let (W,f) be a neural network over Q and let (x,t) be a data sample. If $\left(\mathtt{W}_x^f\right)_\epsilon = 0$, then any other quiver representation V of the delooped quiver Q° that is isomorphic to $\mathtt{W}_x^f$ has $V_\epsilon = 0$. Therefore, if in a dataset $\{(x_i,t_i)\}_{i=1}^N$ the majority of samples (x,t) such that for a specific edge $\epsilon \in Q^\circ$ the corresponding weight on $\mathtt{W}_x^f$ is zero, then the coordinates of $\left[\mathtt{W}_x^f\right]$ inside the moduli space corresponding to ϵ are not used for computations. Therefore, a projection of those coordinates to zero corresponds to the notion of pruning of neural networks, that is forcing to zero the smaller weights on a network [28]. From Equation (5) in page 19, we can see that this interpretation of the data explains why naive pruning works. Namely, if one of the weights in the neural network (W,f) is small, then so does the corresponding weight in $\mathtt{W}_x^f$ for any input x. Since the coordinates of $\mathtt{W}_x^f$ are given in function of the weights of (W,f), by Equation (5) in page 23 and the previous consequence, a small weight of (W,f) sends inputs x to representations $\mathtt{W}_x^f$ with some coordinates equal to zero in the moduli space. If this happens for a big proportion of the samples in the dataset, then the network (W,f) is not using all of the coordinates in the moduli space to represent its data in the form of the map $\varphi(W,f) : \mathbb{C}^d \to_d \mathcal{M}_k(\widetilde{Q})$.

7.1.3. Consequence 3

Let $\mathcal{M}$ be the data manifold in the input space of a neural network (W,f). The map $\varphi(W,f)$ takes $\mathcal{M} \subset \mathbb{C}^d$ to $\varphi(W,f)(\mathcal{M}) \subset {}_d\mathcal{M}_k(\widetilde{Q})$. The subset $\varphi(W,f)(\mathcal{M})$ generates a

sub-manifold of the moduli space (as it is well known in topology [46]) that parametrizes all possible outputs that the neural network (W, f) can produce from inputs on the data manifold $\mathcal{M}$. This means that the geometry of the data manifold $\mathcal{M}$ has been translated into the moduli space ${}_d\mathcal{M}_k(\widetilde{Q})$, and this implies that the mathematical knowledge [25–27] that we have of the geometry of the moduli spaces ${}_d\mathcal{M}_k(\widetilde{Q})$ can be used to understand the dynamics of neural network training, due to the universality of the description of neural networks we have provided.

7.2. Induced Inquiries for Future Research

7.2.1. Inquiry 1

Following Consequence 1, one would like to look for correlations between the dimension of the moduli space and properties of neural networks. The dimension of the moduli space is equal to the number of basis paths in ReLU networks found by Zheng et al. [24], where they empirically confirm that it is a good measure for generalization. This number was also obtained as the rank of a structure matrix for paths in a ReLU network [13], however, they put restrictions on the architecture of the network to compute it. As we noted before, the network function of any neural network passes through the moduli space, where the data quiver representations $\mathtt{W}_x^f$ lie, so the dimension of the moduli space could be used to quantify the capacity of neural networks in general.

7.2.2. Inquiry 2

We can use the moduli space to formulate what training does to the data quiver representations. Training a neural network through gradient descent generates an iterative sequence of neural networks $(W_1, f), (W_2, f), \ldots, (W_m, f)$ where m is the total number of training iterations. For each gradient descent iteration $i = 1, \ldots, m$ we have

$$Im\big(\varphi(W_i, f)\big) \ \subset \ {}_d\mathcal{M}_k(\widetilde{Q}).$$

The moduli space is given only in terms of the combinatorial architecture of the neural network, while the weight and activation architectures determine how the points $\left[\mathtt{W}_{x_1}^f\right], \ldots, \left[\mathtt{W}_{x_n}^f\right]$ are distributed inside the moduli space ${}_d\mathcal{M}_k(\widetilde{Q})$, because of Equation (5). Since the training changes the weights and not (always) the network quiver (unless of course in neural architecture search), we obtain that each training step defines a different map $\varphi(W_i, f) : \mathbb{C}^d \to_d \mathcal{M}_k(\widetilde{Q})$. Therefore, the sub-manifold $Im\big(\varphi(W_i, f)\big)$ is changing its shape during training inside the moduli space ${}_d\mathcal{M}_k(\widetilde{Q})$.

A training of a neural network, which is a sequence of neural networks $(W_1, f), \ldots, (W_m, f)$, can be thought as, first adjusting the manifold $Im\big(\varphi(W_1, f)\big)$ into $Im\big(\varphi(W_2, f)\big)$, then the manifold $Im\big(\varphi(W_2, f)\big)$ into $Im\big(\varphi(W_3, f)\big)$, and so on. This is a completely new way of representing the training of neural networks that works universally for any neural network, which leads to the following question:

"Can training dynamics be made more explicit in these moduli spaces in such a way that allows proving more precise convergence theorems than the currently known?"

7.2.3. Inquiry 3

A training of the form $(W_1, f), \ldots, (W_m, f)$ only changes the weights of the neural network. As we can see, our data quiver representations depend on both the weights and the activations, and therefore a usual training does not exploit completely the fact that the data quiver representations are mapped via φ to the moduli space. Thus, the idea of learning the activation functions, as it is done by Goyal et al. [29], will produce a training of the form $(W_1, f_1), \ldots, (W_m, f_m)$, and this allows the maps $\varphi(W_i, f_i)$ to explore more freely the moduli space than the case where only the weights are learned. Our results imply that a training that changes (and not necessarily learns) the activation functions has the possibility of exploring more the moduli space due to the dependence of the map $\varphi(W, f)$ on the activation functions. One would like to see if this can actually improve the

training of neural networks, and these are exactly the results obtained by the experiments of Goyal et al. [29]. Therefore, the following question arises naturally:

"Can neural network learning be improved by changing activation functions during training?"

8. Conclusions and Future Works

We presented the theoretical foundations for a different understanding of neural networks using their combinatorial and algebraic nature, while explaining current intuitions in deep learning by relying only on the mathematical consequences of the computations of the network during inference. We may summarize our work with the following six points:

1. We use quiver representation theory to represent neural networks and their data processing;
2. This representation of neural networks scales to modern deep architectures like conv layers, pooling layers, residual layers, batch normalization and even randomly wired neural networks [32];
3. Theorem 1 shows that neural networks are algebraic objects, in the sense that the maps preserving the algebraic structure also preserve the computations of the network. Even more, we show that positive scale invariance of ReLU networks is a particular case of this result;
4. We represented data as thin quiver representations with identity activations in terms of the architecture of the network. We proved that this representation of data is algebraically consistent (invariant under isomorphisms) and carries the important notion of feature spaces of all layers at the same time;
5. We introduced the moduli space of a neural network, and proved that it contains all possible (isomorphism classes of) thin quiver representations that result from the computations of the neural network on a forward pass. This leads us to the mathematical formalization of a modified version of the manifold hypothesis in machine learning, given in terms of the architecture of the network;
6. Our representation of neural networks and the data they process is the first to universally represent neural networks: it does not depends on the chosen architecture, activation functions, data, loss function, or even the task.

To the knowledge of the authors, the insights, concepts and results in this work are the first of their kind. In the future, we aim to translate more deep learning objects into the quiver representation language. For instance,

- Dropout [47] is a restriction of the training to several network sub-quivers. This translates into adjustments of the configuration of the data inside the moduli space via sub-spaces given by sub-quivers;
- Generative adversarial networks [48] and actor-critics [49] provide the stage for the interplay between two moduli spaces that get glued together to form a bigger one;
- Recurrent neural networks [50] become a stack of the same network quiver, and therefore the same moduli space gets glued with copies of itself multiple times;
- The knowledge stored in the moduli space in the form of the map $\varphi(W, f)$ provides a new concept to express and understand transfer learning [51]. Extending a trained network will globally change the moduli space, while fixing the map $\varphi(W, f)$ in the sub-space corresponding to the unchanged part of the network quiver.

On expanding the algebraic understanding of neural networks, we consider the following approaches for further research.

- Study the possibility to transfer the gradient descent optimization to the moduli space with the goal of not only optimizing the network weights but also the activation functions.
- The combinatorics of network quivers seem key to the understanding of neural networks and their moduli spaces. Therefore, further study of network quivers by themselves is required [9,10,16].

- Continuity and differentiability of the network function $\Psi(W, f)$ and the map $\varphi(W, f)$ will allow the use of more specific algebraic-geometric tools [25,52]. Even more, the moduli space is a toric variety and then we can use toric geometry [53] to study the moduli space see [54,55].
- Neural networks define finite-dimensional representations of wild hereditary finite-dimensional associative algebras, which can be studied with algebraic-combinatorial techniques [9,56,57].

Finally, this work provides a language in which to state a different kind of scientific hypotheses in deep learning, and we plan to use it as such. Many characterizations will arise from the interplay of algebraic methods and optimization. Namely, when solving a task in practical deep learning, one tries different hidden quivers and optimization hyper-parameters. Therefore, measuring changes in the hidden quiver will become important.

Author Contributions: Conceptualization, P.-M.J.; Formal analysis, M.A.; Supervision, P.-M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

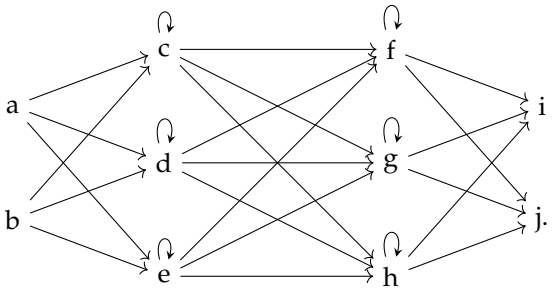
Data Availability Statement: Not applicable.

Acknowledgments: The first named author would like to thank Ibrahim Assem, Thomas Brüstle and Shiping Liu for the provided freedom of research. To Bernhard Keller and Markus Reineke for very useful mathematical exchange. We specially thank Thomas Brüstle and Markus Reineke for their help on computing the dimension of the moduli space provided in Appendix C. This paper was written while the first named author was a postdoctoral fellow at the Université de Sherbrooke.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Example of Theorem 1

Here we illustrate with an example the result of Theorem 1, i.e., that an isomorphism between two neural networks (W, f) and (V, g) preserves their network function $\Psi(W, f)(x) = \Psi(V, g)(x)$. Let us consider a ReLU multilayer perceptron (W, f) with 2 hidden layers of 3 neurons each, 2 neurons on the input layer and 2 neurons on the output layer. That is,



We denote by W_1, W_2 and W_3 the weight matrices of the network from left to right. Consider the explicit matrices

$$W_1 = \begin{pmatrix} 0.2 & -0.4 \\ -1.1 & 1.0 \\ -0.1 & -0.2 \end{pmatrix}, \quad W_2 = \begin{pmatrix} -0.6 & -0.2 & -0.3 \\ 0.3 & 1.2 & -0.4 \\ -0.1 & -1.0 & 0.2 \end{pmatrix}, \quad W_3 = \begin{pmatrix} 0.5 & -0.7 & 0.3 \\ -1.2 & 0.1 & -0.6 \end{pmatrix}.$$

Assume now that the input vector is the vector $x = \begin{pmatrix} -1.2 \\ 0.3 \end{pmatrix}$, then the output of the first layer is

$$ReLU(W_1 x) = ReLU\begin{pmatrix} 0.2(-1.2) - 0.4(0.3) \\ -1.1(-1.2) + 1.0(0.3) \\ -0.1(-1.2) - 0.2(0.3) \end{pmatrix} = ReLU\begin{pmatrix} -0.36 \\ 1.62 \\ 0.06 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.62 \\ 0.06 \end{pmatrix}$$

The output of the second layer is

$$\begin{aligned} ReLU\big(W_2(ReLU(W_1 x))\big) &= ReLU\begin{pmatrix} -0.2(1.62) - 0.3(0.06) \\ 1.2(1.62) - 0.4(0.06) \\ -1.0(1.62) + 0.2(0.06) \end{pmatrix} \\ &= ReLU\begin{pmatrix} -0.342 \\ 1.92 \\ -1.608 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 1.92 \\ 0 \end{pmatrix}. \end{aligned}$$

Therefore, the score (or output) of the network is

$$\begin{aligned} \Psi(W, f)(x) &= W_3\Big(ReLU\big(W_2(ReLU(W_1 x))\big)\Big) \\ &= W_3\begin{pmatrix} 0 \\ 1.92 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} -0.7(1.92) \\ 0.1(1.92) \end{pmatrix} \\ &= \begin{pmatrix} -1.344 \\ 0.192 \end{pmatrix}. \end{aligned}$$

Here we have computed the network function as a sequence of linear maps followed by point-wise non-linearities. Let us remark that our definition of network function is equivalent to this one since following Definition 15 we have

$$\begin{aligned} &\mathbf{a}(W, f)_a(x) = -1.2, \\ &\mathbf{a}(W, f)_b(x) = 0.3, \\ &\mathbf{a}(W, f)_c(x) = ReLU(0.2(-1.2) - 0.4(0.3)) = ReLU(-0.36) = 0, \\ &\mathbf{a}(W, f)_d(x) = ReLU(-1.1(-1.2) + 1.0(0.3)) = ReLU(1.62) = 1.62, \\ &\mathbf{a}(W, f)_e(x) = ReLU(-0.1(-1.2) - 0.2(0.3)) = ReLU(0.06) = 0.06, \\ &\mathbf{a}(W, f)_f(x) = ReLU(-0.6(0) - 0.2(1.62) - 0.3(0.06)) = ReLU(-0.342) = 0, \\ &\mathbf{a}(W, f)_g(x) = ReLU(0.3(0) + 1.2(1.62) - 0.4(0.06)) = 1.92, \\ &\mathbf{a}(W, f)_h(x) = ReLU(-0.1(0) - 1.0(1.62) + 0.2(0.06)) = ReLU(-1.608) = 0, \\ &\mathbf{a}(W, f)_i(x) = 0.5(0) - 0.7(1.92) + 0.3(0) = -1.344, \\ &\mathbf{a}(W, f)_j(x) = -1.2(0) + 0.1(1.92) - 0.6(0) = 0.192. \end{aligned}$$

Consider now a change of basis for (W, f). As mentioned in the text, the change of basis for the input and output neurons are set to 1 (i.e., $\tau_a = \tau_b = \tau_i = \tau_j = 1$). As for the six

hidden neurons, let us consider the following six change of basis $\tau = \begin{pmatrix} -0.2 & 1.0 \\ 0.3 & -1.0 \\ -1.1 & 0.1 \end{pmatrix}$. This change of basis can be applied to the weights following Equation (1). Since τ is 1 for the input neurons, the weights of the first layer get transformed as follows

$$\tau W_1 = \begin{pmatrix} 0.2(-0.2) & -0.4(-0.2) \\ -1.1(0.3) & 1.0(0.3) \\ -0.1(-1.1) & -0.2(-1.1) \end{pmatrix} = \begin{pmatrix} -0.04 & 0.08 \\ -0.33 & 0.3 \\ 0.11 & 0.22 \end{pmatrix}.$$

The weights of the second layer become

$$\tau W_2 = \begin{pmatrix} -0.6\frac{1.0}{-0.2} & -0.2\frac{1.0}{0.3} & -0.3\frac{1.0}{-1.1} \\ 0.3\frac{-1.0}{-0.2} & 1.2\frac{-1.0}{0.3} & -0.4\frac{-1.0}{-1.1} \\ -0.1\frac{0.1}{-0.2} & -1.0\frac{0.1}{0.3} & 0.2\frac{0.1}{-1.1} \end{pmatrix} = \begin{pmatrix} 3 & -\frac{0.2}{0.3} & \frac{0.3}{1.1} \\ \frac{0.3}{0.2} & -4 & -\frac{0.4}{1.1} \\ 0.05 & -\frac{0.1}{0.3} & -\frac{0.02}{1.1} \end{pmatrix},$$

and those of the third layer

$$\tau W_3 = \begin{pmatrix} \frac{0.5}{1.0} & \frac{-0.7}{-1.0} & \frac{0.3}{0.1} \\ \frac{-1.2}{1.0} & \frac{0.1}{-1.0} & \frac{-0.6}{0.1} \end{pmatrix} = \begin{pmatrix} 0.5 & 0.7 & 3 \\ -1.2 & -0.1 & -6 \end{pmatrix}.$$

As for the activations, one has to apply Equation (2), i.e., $\tau_v ReLU(\frac{x}{\tau_v})$ in our case. Note that if $\tau_v > 0$ then $\tau_v ReLU(\frac{x}{\tau_v}) = \frac{\tau_v}{\tau_v} ReLU(x) = ReLU(x)$ which derives from the positive scale invariance of ReLU. However, if $\tau_v < 0$ then $\tau_v ReLU(\frac{x}{\tau_v}) = min(x, 0) = g(x)$. Considering the change of basis matrix τ given before, it derives that $\tau f = \begin{pmatrix} g & ReLU \\ ReLU & g \\ g & ReLU \end{pmatrix}$.

Let us apply a forward pass on the neural network $(\tau W, \tau f)$ for the same input $x = \begin{pmatrix} -1.2 \\ 0.3 \end{pmatrix}$. On the first layer we have:

$$\begin{aligned} \begin{pmatrix} g \\ ReLU \\ g \end{pmatrix} \tau W_1 x &= \begin{pmatrix} g \\ ReLU \\ g \end{pmatrix} \begin{pmatrix} -0.04(-1.2) + 0.08(0.3) \\ -0.33(-1.2) + 0.3(0.3) \\ 0.11(-1.2) + 0.22(0.3) \end{pmatrix} \\ &= \begin{pmatrix} g \\ ReLU \\ g \end{pmatrix} \begin{pmatrix} 0.072 \\ 0.486 \\ -0.066 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 0.486 \\ -0.066 \end{pmatrix}. \end{aligned}$$

Therefore, the activation output of the neurons in the first hidden layer is equal to the activation output on the same neurons on the neural network (W, f) times the corresponding change of basis.

Propagating the signal to the other layer leads to

$$\tau W_2 \begin{pmatrix} 0 \\ 0.486 \\ -0.066 \end{pmatrix} = \begin{pmatrix} -\frac{0.2}{0.3}(0.486) + \frac{0.3}{1.1}(-0.066) \\ -4(0.486) - \frac{0.4}{1.1}(-0.066) \\ -\frac{0.1}{0.3}(0.486) - \frac{0.02}{1.1}(-0.066) \end{pmatrix} = \begin{pmatrix} -0.342 \\ -1.92 \\ -0.1608 \end{pmatrix},$$

and after applying the activation $\begin{pmatrix} ReLU \\ g \\ ReLU \end{pmatrix}$ we get the vector $\begin{pmatrix} 0 \\ -1.92 \\ 0 \end{pmatrix}$. Finally,

$$\Psi\Big((\tau W, \tau f)\Big)(x) = \tau W_3 \begin{pmatrix} 0 \\ -1.92 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.7(-1.92) \\ -0.1(-1.92) \end{pmatrix} = \begin{pmatrix} -1.344 \\ 0.192 \end{pmatrix},$$

which is the same output as the one for (W, f) computed before. We can also observe that the activation output of each hidden (and output) neuron on $(\tau W, \tau f)$ is equal to the activation output on that same neuron in (W, f) times the change of basis of that neuron, as noted in the proof of Theorem 1.

Appendix B. Example of Theorem 2

Here we compute an example to illustrate that $\Psi(W, f)(x) = \Psi(\mathtt{W}_x^f, 1)(1^d)$. We will work with the notation of Appendix A for the ReLU MLP with explicit weight matrices W_1, W_2 and W_3 and input vector x given by

$$W_1 = \begin{pmatrix} 0.2 & -0.4 \\ -1.1 & 1.0 \\ -0.1 & -0.2 \end{pmatrix}, \qquad W_2 = \begin{pmatrix} -0.6 & -0.2 & -0.3 \\ 0.3 & 1.2 & -0.4 \\ -0.1 & -1.0 & 0.2 \end{pmatrix},$$

$$W_3 = \begin{pmatrix} 0.5 & -0.7 & 0.3 \\ -1.2 & 0.1 & -0.6 \end{pmatrix}, \qquad x = \begin{pmatrix} -1.2 \\ 0.3 \end{pmatrix}.$$

Recall the definition of the representation $\mathtt{W}_x^f$,

$$\left(\mathtt{W}_x^f\right)_\epsilon = \begin{cases} W_\epsilon x_{s(\epsilon)} & \text{if } s(\epsilon) \text{ is an input vertex,} \\ W_\epsilon & \text{if } s(\epsilon) \text{ is a bias vertex,} \\ W_\epsilon \dfrac{\mathbf{a}(W,f)_{s(\epsilon)}(x)}{\sum\limits_{\beta \in \zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W,f)_{s(\beta)}(x)} & \text{if } s(\epsilon) \text{ is a hidden vertex.} \end{cases}$$

Denote by V_1, V_2 and V_3 the weight matrices of $\mathtt{W}_x^f$. We can easily see that V_1 is given by

$$V_1 = \begin{pmatrix} 0.2(-1.2) & -0.4(0.3) \\ -1.1(-1.2) & 1.0(0.3) \\ -0.1(-1.2) & -0.2(0.3) \end{pmatrix} = \begin{pmatrix} -0.24 & -0.12 \\ 1.32 & 0.3 \\ 0.12 & -0.06 \end{pmatrix},$$

As for the next weight matrices, we have already computed the pre-activations and post-activations of each neuron in a forward pass of x through the network $(W, ReLU)$ in the last appendix, then

$$V_2 = \begin{pmatrix} -0.6\frac{0}{-0.36} & -0.2\frac{1.62}{1.62} & -0.3\frac{0.06}{0.06} \\ 0.3\frac{0}{-0.36} & 1.2\frac{1.62}{1.62} & -0.4\frac{0.06}{0.06} \\ -0.1\frac{0}{-0.36} & -1.0\frac{1.62}{1.62} & 0.2\frac{0.06}{0.06} \end{pmatrix} = \begin{pmatrix} 0 & -0.2 & -0.3 \\ 0 & 1.2 & -0.4 \\ 0 & -1.0 & 0.2 \end{pmatrix},$$

and also

$$V_3 = \begin{pmatrix} 0.5\frac{0}{-0.342} & -0.7\frac{1.92}{1.92} & 0.3\frac{0}{-1.608} \\ -1.2\frac{0}{-0.342} & 0.1\frac{1.92}{1.92} & -0.6\frac{0}{-1.608} \end{pmatrix} = \begin{pmatrix} 0 & -0.7 & 0 \\ 0 & 0.1 & 0 \end{pmatrix}.$$

Let us compute a forward pass of the network $(V, 1) = (\mathtt{W}_x^f, 1)$ given the input $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and verify that the output is the same as that of Appendix A. We have that

$$\begin{aligned} V_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} &= \begin{pmatrix} -0.24 - 0.12 \\ 1.32 + 0.3 \\ 0.12 - 0.06 \end{pmatrix} \\ &= \begin{pmatrix} -0.36 \\ 1.62 \\ 0.06 \end{pmatrix}, \end{aligned}$$

and

$$\begin{aligned} V_2 \begin{pmatrix} -0.36 \\ 1.62 \\ 0.06 \end{pmatrix} &= \begin{pmatrix} -0.2(1.62) - 0.3(0.06) \\ 1.2(1.62) - 0.4(0.06) \\ -1.0(1.62) + 0.2(0.06) \end{pmatrix} \\ &= \begin{pmatrix} -0.342 \\ 1.92 \\ -1.608 \end{pmatrix}, \end{aligned}$$

and finally,

$$\begin{aligned} V_3 \begin{pmatrix} -0.342 \\ 1.92 \\ -1.608 \end{pmatrix} &= \begin{pmatrix} -0.7(1.92) \\ 0.1(1.92) \end{pmatrix} \\ &= \begin{pmatrix} -1.344 \\ 0.192 \end{pmatrix}. \end{aligned}$$

As noted in the proof of Theorem 2, the activation output of each neuron in $(\mathtt{W}_x^f, 1)$ after a forward pass of the input vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, is equal to the pre-activation of that same neuron in $(W, ReLU)$ after a forward pass of x.

Appendix C. Double-Framed Quiver Moduli

Let $Q = (\mathcal{V}, \mathcal{E}, s, t)$ be a network quiver. Recall that the delooped quiver $Q^\circ = (\mathcal{V}^\circ, \mathcal{E}^\circ, s^\circ, t^\circ)$ is obtained from Q by removing all loops. The hidden quiver $\widetilde{Q} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{s}, \widetilde{t})$ is obtained from the delooped quiver by removing the input and the output layers. Once the complex vector spaces $\mathbb{C}$ are fixed to every vertex of Q, the space of stable thin representations of Q° is

$$\mathcal{R} := \{W : W_\alpha \in \mathbb{C}^* \text{ for every } \alpha \in \mathcal{E}^\circ\}.$$

This is an affine space isomorphic to $\mathcal{R} \cong \mathbb{C}^n$, where n is the number of elements of $\mathcal{E}^\circ$. The change of basis group we consider here is the group

$$\widetilde{G} := \prod_{v \in \widetilde{\mathcal{V}}} \mathbb{C}^*,$$

whose action on $\mathcal{R}$ is given by Equation (1) in page 6, i.e., $(\tau \cdot W)_\alpha = \tau_{s(\alpha)}^{-1} W_\alpha \tau_{t(\alpha)}$. The action (Definition 8) of the group $\widetilde{G}$ is **free** if given $g, h \in \widetilde{G}$ the existence of an element x with $g \cdot x = h \cdot x$ implies that $g = h$.

Lemma A1. *The action of the group $\widetilde{G}$ is free.*

Proof. Let $\tau = (\tau_v)_{v \in \widetilde{\mathcal{V}}} \in \widetilde{G}$ be an element different from the identity, that is, there is $v \in \widetilde{\mathcal{V}}$ such that $\tau_v \neq 1$, and let W be a thin representation. Since v is a vertex of the hidden quiver, there exists an edge α with target v and source $s(\alpha) = u$. If u is a source vertex, then the weight of $\tau \cdot W$ in the edge α is $W_\alpha \tau_v$, which is different from W_α since $W_\alpha \neq 0$. If u is not a source, then the weight of $g \cdot W$ in the edge α is $\tau_u^{-1} W_\alpha \tau_v$ which is different from W_α unless $\tau_u = \tau_v$. We can apply the same argument to the vertex u until we reach a source of Q, which shows that $\tau \cdot W \neq W$ for any τ that is not the identity of $\widetilde{G}$. This is equivalent to the action of the group being free [30]. □

Given the action of a group G on a set X, the G**-orbit** of an element $x \in X$ is by definition the set $\{g \cdot x \in X : g \in G\}$. In our case, the $\widetilde{G}$-orbit of a thin representation W is the set

$$\{\tau \cdot W : \tau \in \widetilde{G}\},$$

which is the isomorphism class of the representation W, i.e., the set of all representations isomorphic to W. From Section 7 we obtain that the moduli space is the set of all $\widetilde{G}$-orbits of elements in $\mathcal{R}$. We will use this to prove the following:

Theorem A1. *The dimension of the moduli space is*

$$dim_{\mathbb{C}}\left({}_d\mathcal{M}_k(\widetilde{Q})\right) = \#\mathcal{E}^\circ - \#\widetilde{\mathcal{V}}.$$

Proof. Let W be a thin representation of Q° with non-zero weights. For every hidden vertex $v \in \widetilde{\mathcal{V}}$ we are going to choose once and for all an oriented edge, that we denote by $\alpha_v : v' \to v$. The collection of all the chosen edges α_v and vertices that are targets and sources of them form a subquiver of Q°, that we denote $Q^\vee$. The number of hidden vertices of Q and the number of edges in the quiver $Q^\vee$ are the same, by construction. Moreover, there cannot be non-oriented cycles in the quiver $Q^\vee$ since we would have to had chosen two oriented edges with the same target, and we are only choosing one in our construction. This implies that $Q^\vee$ is a union of trees, and the intersection of any two of those trees can only be a source vertex of Q by the same argument. Furthermore, for any of those trees the only vertex that is not a hidden vertex is a unique source of Q corresponding to that tree.

We will show that we can choose a change of basis for each of these trees so that all its weights can be set to 1 in the isomorphism class of W, i.e., the $\widetilde{G}$-orbit of that representation. Once this is done, we only have to count how many free choices we have left for weights of Q° that have not been set to 1. This is exactly the number of oriented edges of Q° minus the

number of hidden vertices (which are in correspondence with the oriented edges forming the trees in $Q^\vee$). This will be the dimension of the space of $\widetilde{G}$-orbits (i.e., the moduli space) because the previous lemma implies that this is indeed the minimum number of weights to describe the representation W up to isomorphisms.

Let T be a tree in $Q^\vee$ and let v be its source vertex. A change of basis $\tau \in \widetilde{G}$ has $\tau_v = 1$. Let $\alpha_1 : v \to v_1$ be an oriented edge of T with source v, and take $\tau_{v_1} = \dfrac{1}{W_{\alpha_1}}$, so after applying τ to W we obtain that $(\tau \cdot W)_{\alpha_1} = 1$. Let $\alpha_2 : v_1 \to v_2$ be an oriented edge in T that starts in v_1, and take $\tau_{v_2} = \dfrac{1}{W_{\alpha_1} W_{\alpha_2}}$. After applying τ we obtain $(\tau \cdot W)_{\alpha_2} = 1$. An induction argument shows that up to isomorphism we can take all the weights of W along the tree T to be equal to 1. Finally, the trees in $Q^\vee$ do not share any oriented edges, therefore, they also do not share any vertices except for the source vertices, for which the change of basis is set to 1. This means that we have chosen a change of basis for every hidden vertex so that the resulting isomorphic representation has all its weights along the trees of $Q^\vee$ equal to 1, which completes the proof. □

Appendix D. Glossary

Here we gather all definitions given in this paper alphabetically.

Activation function. An **activation function** is a one-variable non-linear function $f : \mathbb{C} \to \mathbb{C}$ differentiable except in a set of measure zero.

Activation output of vertices/neurons. Let (W, f) be a neural network over a network quiver Q and let $x \in \mathbb{C}^d$ be an input vector of the network. Denote by ζ_v the set of edges of Q with target v. The **activation output of the vertex** $v \in \mathcal{V}$ **with respect to** x after applying a forward pass is denoted $\mathbf{a}(W, f)_v(x)$ and is computed as follows:

- If $v \in \mathcal{V}$ is an input vertex, then $\mathbf{a}(W, f)_v(x) = x_v$;
- If $v \in \mathcal{V}$ is a bias vertex, then $\mathbf{a}(W, f)_v(x) = 1$;
- If $v \in \mathcal{V}$ is a hidden vertex, then $\mathbf{a}(W, f)_v(x) = f_v\left(\sum_{\alpha \in \zeta_v} W_\alpha \mathbf{a}(W, f)_{s(\alpha)}(x)\right)$;
- If $v \in \mathcal{V}$ is an output vertex, then $\mathbf{a}(W, f)_v(x) = \sum_{\alpha \in \zeta_v} W_\alpha \mathbf{a}(W, f)_{s(\alpha)}(x)$;
- If $v \in \mathcal{V}$ is a max-pooling vertex, then $\mathbf{a}(W, f)_v(x) = \mathbf{max}_\alpha \, Re\big(W_\alpha \mathbf{a}(W, f)_{s(\alpha)}(x)\big)$, where Re denotes the real part of a complex number, and the maximum is taken over all $\alpha \in \mathcal{E}$ such that $t(\alpha) = v$.

Architecture of a neural network (Ref. [3], p. 193). The **architecture** of a neural network refers to its structure which accounts for how many units (neurons) it has and how these units are connected together.

Change of basis group of thin representations. The **change of basis group** of thin representations over a quiver Q is

$$G = \prod_{v \in \mathcal{V}} \mathbb{C}^*,$$

where $\mathbb{C}^*$ denotes the multiplicative group of non-zero complex numbers. That is, the elements of G are vectors of non-zero complex numbers $\tau = (\tau_1, \ldots, \tau_n)$ indexed by the set $\mathcal{V}$ of vertices of Q, and the group operation between two elements $\tau = (\tau_1, \ldots, \tau_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_n)$ is by definition

$$\tau\sigma := (\tau_1\sigma_1, \ldots, \tau_n\sigma_n).$$

Change of basis group of double-framed thin quiver representations. The group of **change of basis of double-framed thin quiver representations** is the same group $\widetilde{G}$ of change of basis of neural networks.

Change of basis group of neural networks. The **group of change of basis** for neural networks is denoted as

$$\widetilde{G} = \prod_{v \in \widetilde{\mathcal{V}}} \mathbb{C}^*.$$

An element of the change of basis group $\widetilde{G}$ is called a **change of basis** of the neural network (W, f).

Co-framed quiver representation (Ref. [25]). A choice of a thin representation $\widetilde{W}$ of the hidden quiver and a map $\ell_v : U_v \to \widetilde{W}_v$ for each $v \in \widetilde{\mathcal{V}}$ determines a pair $(\widetilde{W}, \ell)$, where $\ell = \{\ell_v\}_{v \in \widetilde{\mathcal{V}}}$ which is known as a **co-framed** quiver representation of $\widetilde{Q}$ by the family of vector spaces $\{U_v\}_{v \in \widetilde{\mathcal{V}}}$.

Combinatorial/weight/activation architectures. The **combinatorial architecture** of a neural network is its network quiver. The **weight architecture** is given by constraints on how the weights are chosen, and the **activation architecture** is the set of activation functions assigned to the loops of the network quiver.

Data quiver representations. Let (W, f) be a neural network over the network quiver Q and $x \in \mathbb{C}^d$ an input vector. The thin quiver representation $\mathtt{W}_x^f$ is defined as

$$\left(\mathtt{W}_x^f\right)_\epsilon = \begin{cases} W_\epsilon x_{s(\epsilon)} & \text{if } s(\epsilon) \text{ is an input vertex,} \\ W_\epsilon & \text{if } s(\epsilon) \text{ is a bias vertex,} \\ W_\epsilon \dfrac{\mathbf{a}(W,f)_{s(\epsilon)}(x)}{\sum\limits_{\beta \in \zeta_{s(\epsilon)}} W_\beta \cdot \mathbf{a}(W,f)_{s(\beta)}(x)} & \text{if } s(\epsilon) \text{ is a hidden vertex,} \end{cases}$$

Delooped quiver of a network quiver. The **delooped** quiver Q° of Q is the quiver obtained by removing all loops of Q. We denote $Q^\circ = (\mathcal{V}, \mathcal{E}^\circ, s^\circ, t^\circ)$.

Double-framed thin quiver representation. A **double-framed** thin quiver representation is a triple $(\ell, \widetilde{W}, h)$ where $\widetilde{W}$ is a thin quiver representation of the hidden quiver, $(\widetilde{W}, h)$ is a framed representation of $\widetilde{Q}$ and $(\widetilde{W}, \ell)$ is a co-framed representation of $\widetilde{Q}$.

Framed quiver representation (Ref. [25]). A choice of a thin representation $\widetilde{W}$ of the hidden quiver and a map $h_v : \widetilde{W}_v \to V_v$ for each $v \in \widetilde{\mathcal{V}}$ determines a pair $(\widetilde{W}, h)$, where $h = \{h_v\}_{v \in \widetilde{\mathcal{V}}}$, that is known as a **framed** quiver representation of $\widetilde{Q}$ by the family of vector spaces $\{V_v\}_{v \in \widetilde{\mathcal{V}}}$.

Group (Ref. [30], Chapter 1). A non-empty set G is called a **group** if there exists a function $\cdot : G \times G \to G$, called the product of the group denoted $a \cdot b$, such that

- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$.
- There exists an element $e \in G$ such that $e \cdot a = a \cdot e = a$ for all $a \in G$, called the **identity** of G.
- For each $a \in G$ there exists $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Group action (Ref. [30], Chapter 3). Let G be a group and let X be a set. We say that there is an **action of** G **on** X if there exists a map $\cdot : G \times X \to X$ such that

- $e \cdot x = x$ for all $x \in X$, where $e \in G$ is the identity.
- $a \cdot (b \cdot x) = (ab) \cdot x$, for all $a, b \in G$ and all $x \in X$.

Hidden quiver. The **hidden quiver** of Q, denoted by $\widetilde{Q} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{s}, \widetilde{t})$, is given by the hidden vertices $\widetilde{\mathcal{V}}$ of Q and all the oriented edges $\widetilde{\mathcal{E}}$ between hidden vertices of Q that are not loops.

Input/Output vertices. We call **input vertices** of the hidden quiver $\widetilde{Q}$ the vertices that are connected to the input vertices of Q, and we call **output vertices** of the hidden quiver $\widetilde{Q}$ the vertices that are connected to the output vertices of Q.

Isomorphic quiver representations. Let Q be a quiver and let W and U be two representations of Q. If there is a morphism of representations $\tau : W \to U$ where each τ_v is an invertible linear map, then W and U are said to be **isomorphic representations**.

Labeled data set. A **labeled data set** is given by a finite set $D = \{(x_i, t_i)\}_{i=1}^{n}$ of pairs such that $x_i \in \mathbb{C}^d$ is a data vector (could also be a matrix or a tensor) and t_i is a target. We can have $t_i \in \mathbb{C}^k$ for a regression and $t_i \in \{C_0, C_1, \ldots, C_k\}$ for a classification.

Moduli space. The **moduli space** of stable double-framed thin quiver representations of $\widetilde{Q}$ is by definition

$$ {}_d\mathcal{M}_k(\widetilde{Q}) := \big\{[V] : V \in {}_d\mathcal{R}_k(\widetilde{Q}) \text{ is stable}\big\}. $$

Morphism/Isomorphism of neural networks. Let (W, f) and (V, g) be neural networks over the same network quiver Q. A **morphism of neural networks** $\tau : (W, f) \to (V, g)$ is a morphism of thin quiver representations $\tau : W \to V$ such that $\tau_v = 1$ for all $v \in \mathcal{V}$ that is not a hidden vertex, and for every hidden vertex $v \in \mathcal{V}$ the following diagram is commutative.

$$ \begin{array}{ccc} \mathbb{C} & \xrightarrow{f_v} & \mathbb{C} \\ {\scriptstyle \tau_v}\downarrow & & \downarrow{\scriptstyle \tau_v} \\ \mathbb{C} & \xrightarrow[g_v]{} & \mathbb{C}. \end{array} $$

A morphism of neural networks $\tau : (W, f) \to (V, g)$ is an **isomorphism of neural networks** if $\tau : W \to V$ is an isomorphism of quiver representations. We say that two neural networks over Q are **isomorphic** if there exists an isomorphism of neural networks between them.

Morphism of quiver representations (Ref. [9], Chapter 3). Let Q be a quiver and let W and U be two representations of Q. A **morphism of representations** $\tau : W \to U$ is a set of linear maps $\tau = (\tau_v)_{v \in \mathcal{V}}$ indexed by the vertices of Q, where $\tau_v : W_v \to U_v$ is a linear map such that $\tau_{t(\epsilon)} W_\epsilon = U_\epsilon \tau_{s(\epsilon)}$ for every $\epsilon \in \mathcal{E}$.

Network function. Let (W, f) be a neural network over a network quiver Q. The **network function** of the neural network is the function

$$ \Psi(W, f) : \mathbb{C}^d \to \mathbb{C}^k $$

where the coordinates of $\Psi(W, f)(x)$ are the activation outputs of the output vertices of (W, f) (often called the "score" of the neural net) with respect to an input vector $x \in \mathbb{C}^d$.

Network quiver. A **network quiver** Q is a quiver arranged by layers such that:

1. There are no loops on source (i.e., input and bias) nor sink vertices;
2. There is exactly one loop on each hidden vertex.

Neural network. A **neural network** over a network quiver Q is a pair (W, f) where W is a thin representation of the delooped quiver Q° and $f = (f_v)_{v \in \mathcal{V}}$ are activation functions, assigned to the loops of Q.

Quiver (Ref. [9], Chapter 2). A **quiver** Q is given by a tuple $(\mathcal{V}, \mathcal{E}, s, t)$ where $(\mathcal{V}, \mathcal{E})$ is an oriented graph with a set of vertices $\mathcal{V}$ and a set of oriented edges $\mathcal{E}$, and maps $s, t : \mathcal{E} \to \mathcal{V}$ that send $\epsilon \in \mathcal{E}$ to its source vertex $s(\epsilon) \in \mathcal{V}$ and target vertex $t(\epsilon) \in \mathcal{V}$, respectively.

Quiver arranged by layers. A quiver Q is **arranged by layers** if it can be drawn from left to right arranging its vertices in columns such that:

- There are no oriented edges from vertices on the right to vertices on the left;
- There are no oriented edges between vertices in the same column, other than loops and edges from bias vertices.

The first layer on the left, called the **input layer**, will be formed by the d input vertices. The last layer on the right, called the **output layer**, will be formed by the k output vertices. The layers that are not input nor output layers are called **hidden layers**. We enumerate the hidden layers from left to right as: 1st hidden layer, 2nd hidden layer, 3rd hidden layer, and so on.

Quiver representation (Ref. [9], Chapter 3). If Q is a quiver, a **quiver representation** of Q is given by a pair of sets

$$W := \left((W_v)_{v\in\mathcal{V}}, (W_\epsilon)_{\epsilon\in\mathcal{E}}\right)$$

where the W_v's are vector spaces indexed by the vertices of Q, and the W_ϵ's are linear maps indexed by the oriented edges of Q, such that for every edge $\epsilon \in \mathcal{E}$:

$$W_\epsilon : W_{s(\epsilon)} \to W_{t(\epsilon)}.$$

Representation space. The **representation space** ${}_d\mathcal{R}_k(\widetilde{Q})$ of the hidden quiver $\widetilde{Q}$ of a network quiver Q, is the set of all possible double-framed thin quiver representations of $\widetilde{Q}$.

Source/Sink vertices (Ref. [9], Chapter 2). A **source vertex** of a quiver Q is a vertex $v \in \mathcal{V}$ such that there are no oriented edges $\epsilon \in \mathcal{E}$ with target $t(\epsilon) = v$. A **sink vertex** of a quiver Q is a vertex $v \in \mathcal{V}$ such that there are no oriented edges $\epsilon \in \mathcal{E}$ with source $s(\epsilon) = v$. A **loop** in a quiver Q is an oriented edge ϵ such that $s(\epsilon) = t(\epsilon)$.

Stable quiver representation. A double-framed thin quiver representation $(\ell, \widetilde{W}, h)$ is **stable** if the following two conditions are satisfied:

1. The only sub-representation U of $\widetilde{W}$ which is contained in $ker(h)$ is the zero sub-representation, and
2. The only sub-representation U of $\widetilde{W}$ that contains $Im(\ell)$ is $\widetilde{W}$.

Sub-representation (Ref. [10], p. 14). Let W be a thin representation of the delooped quiver Q° of a network quiver Q. A **sub-representation** of W is a representation U of Q° such that there is a morphism of representations $\tau : U \to W$ where each map τ_v is an injective map.

Teleportation. Let (W, f) be a neural network and let $\tau \in \widetilde{G}$ be an element of the group of change of basis of neural networks such that the isomorphic neural network $\tau \cdot (W, f)$ has the same weight architecture as (W, f). The **teleportation** of the neural network (W, f) with respect to τ is the neural network $\tau \cdot (W, f)$.

Thin quiver representation. A **thin representation** of a quiver Q is a quiver representation W such that $W_v = \mathbb{C}$ for all $v \in V$.

Zero representation. The **zero representation** of Q is the representation denoted 0 where every vector space assigned to every vertex is the zero vector space, and therefore every linear map in it is also zero.

References

1. Raghu, M.; Schmidt, E. A Survey of Deep Learning for Scientific Discovery. *arXiv* **2020**, arXiv:2003.11755.
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
3. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningBook.org (accessed on 18 November 2021).
4. Rumelhart, D.; Hinton, G.; Williams, R. Learning Representations by Back-Propagating Errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
5. LeCun, Y.; Boser, B.; Denker, J.; Henderson, D.; Howard, R.; Hubbard, W.; Jackel, L. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
6. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the Twenty-sixth Annual Conference on Neural Information Processing Systems (NIPS) , Stateline, NV, USA, 3–8 December 2012; pp. 1097–1105.
7. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
8. Li, H.; Xu, Z.; Taylor, G.; Studer, C.; Goldstein, T. Visualizing the Loss Landscape of Neural Nets. In *Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2018; pp. 6391–6401.
9. Assem, I.; Simson, D.; Skowronski, A. *Elements of the Representation Theory of Associative Algebras. Volume 1: Techniques of Representation Theory*; London Mathematical Society Student Texts; Cambridge University Press: Cambridge, UK, 2006; Volume 65, pp. ix, 458.
10. Schiffler, R. *Quiver Representations*; CMS Books in Mathematics; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. XI, 230.

11. Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef]
12. Dinh, L.; Pascanu, R.; Bengio, S.; Bengio, Y. Sharp Minima Can Generalize for Deep Nets. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1019–1028.
13. Meng, Q.; Zheng, S.; Zhang, H.; Chen, W.; Ye, Q.; Ma, Z.; Yu, N.; Liu, T. G-SGD: Optimizing ReLU Neural Networks in its Positively Scale-Invariant Space. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
14. Neyshabur, B.; Salakhutdinov, R.; Srebro, N. Path-SGD: Path-Normalized Optimization in Deep Neural Networks. In *International Conference on Neural Information Processing Systems—Volume 2*; MIT Press: Cambridge, MA, USA, 2015; pp. 2422–2430.
15. Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G.B.; LeCun, Y. The Loss Surfaces of Multilayer Networks. In Proceedings of the International Conference on Artificial Intelligence and Statistics, San Diego, CA, USA, 9–12 May 2015; Lebanon, G., Vishwanathan, S.V.N., Eds.; ML Research Press: New York, NY, USA, 2015; Volume 38, pp. 192–204.
16. Barot, M. *Introduction to the Representation Theory of Algebras*; Springer International Publishing: Berlin/Heidelberg, Germany, 2015; pp. X, 179.
17. Wood, J.; Shawe-Taylor, J. Representation theory and invariant neural networks. *Discret. Appl. Math.* **1996**, *69*, 33–60. [CrossRef]
18. Healy, M.J.; Caudell, T.P. *Neural Networks, Knowledge, and Cognition: A Mathematical Semantic Model Based upon Category Theory*; Technical Report EECE-TR-04-020; Department of Electrical and Computer Engineering, University of New Mexico: Cambridge, MA, USA, 2004.
19. Chindris, C.; Kline, D. Simultaneous robust subspace recovery and semi-stability of quiver representations. *J. Algebra* **2021**, *577*, 210–236. [CrossRef]
20. Arora, S. ICML 2018: Tutorial Session: Toward the Theoretical Understanding of Deep Learning. Video, 2018. Available online: https://icml.cc/Conferences/2018/Schedule?type=Tutorial (accessed on 15 October 2021).
21. Yosinski, J.; Clune, J.; Nguyen, A.; Fuchs, T.; Lipson, H. Understanding neural networks through deep visualization. In Proceedings of the ICML Deep Learning Workshop, Lille, France, 6–11 July 2015.
22. Feghahati, A.; Shelton, C.R.; Pazzani, M.J.; Tang, K. CDeepEx: Contrastive Deep Explanations. In *ECAI 2020*; IOS: Amsterdam, The Netherlands, 2020; Volume 325, pp. 1143–1151. [CrossRef]
23. Hinton, G.E. Learning multiple layers of representation. *Trends Cognit. Sci.* **2007**, *11*, 428–434. [CrossRef]
24. Zheng, S.; Meng, Q.; Zhang, H.; Chen, W.; Yu, N.; Liu, T. Capacity Control of ReLU Neural Networks by Basis-Path Norm. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; AAAI Press: Palo Alto, California, USA, 2019; pp. 5925–5932.
25. Reineke, M. Framed Quiver Moduli, Cohomology, and Quantum Groups. *J. Algebra* **2008**, *320*, 94–115. [CrossRef]
26. Das, P.; Manikandan, S.; Raghavendra, N. Holomorphic aspects of moduli of representations of quivers. *Indian J. Pure Appl. Math.* **2019**, *50*, 549–595. [CrossRef]
27. Franzen, H.; Reineke, M.; Sabatini, S. Fano quiver moduli. *arXiv* **2020**, arXiv:abs/2001.10556.
28. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In Proceedings of the International Conference on Learning Representations, Louisiana, NO, USA, 6–9 May 2019.
29. Goyal, M.; Goyal, R.; Lall, B. Learning Activation Functions: A New Paradigm of Understanding Neural Networks. *arXiv* **2019**, arXiv:1906.09529.
30. Rotman, J. *An Introduction to the Theory of Groups*; Graduate Texts in Mathematics; Springer: New York, NY, USA, 1995; Volume 148, pp. XV, 517.
31. Nitta, T. An Extension of the Back-Propagation Algorithm to Complex Numbers. *Neural Netw.* **1997**, *10*, 1391–1415. [CrossRef]
32. Xie, S.; Kirillov, A.; Girshick, R.; He, K. Exploring Randomly Wired Neural Networks for Image Recognition. *arXiv* **2019**, arXiv:abs/1904.01569.
33. Wei, T.; Wang, C.; Rui, Y.; Chen, C.W. Network Morphism. In Proceedings of the International Conference on International Conference on Machine Learning—Volume 48, New York, NY, USA, 19–24 June 2016; pp. 564–572.
34. Meng, L.; Zhang, J. IsoNN: Isomorphic Neural Network for Graph Representation Learning and Classification. *arXiv* **2019**, arXiv:1907.09495.
35. Stagge, P.; Igel, C. Neural network structures and isomorphisms: random walk characteristics of the search space. In Proceedings of the 2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks Cat. No.00, San Antonio, TX, USA, 11–13 May 2000; pp. 82–90.
36. Badrinarayanan, V.; Mishra, B.; Cipolla, R. Understanding Symmetries in Deep Networks. *arXiv* **2015**, arXiv:1511.01029.
37. Yi, M.; Meng, Q.; Chen, W.; Ma, Z.; Liu, T. Positively Scale-Invariant Flatness of ReLU Neural Networks. *arXiv* **2019**, arXiv:1903.02237.
38. Yuan, Q.; Xiao, N. Scaling-Based Weight Normalization for Deep Neural Networks. *IEEE Access* **2019**, *7*, 7286–7295. [CrossRef]
39. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
40. Hinton, G. Geoffrey Hinton Talk 'What is Wrong with Convolutional Neural Nets ?'. Video, 2014. Available online: http://www.moreisdifferent.com/2017/09/hinton-whats-wrong-with-CNNs (accessed on 15 October 2021).

41. Sabour, S.; Frosst, N.; Hinton, G.E. Dynamic Routing Between Capsules. *International Conference on Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 3859–3869.
42. Kosiorek, A.; Sabour, S.; Teh, Y.W.; Hinton, G. Stacked Capsule Autoencoders. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32, pp. 15512–15522.
43. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the International Conference on Machine Learning—Volume 37, Lille, France, 7–9 July 2015; pp. 448–456.
44. Akhtar, N.; Mian, A. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access* **2018**, *6*, 14410–14430. [CrossRef]
45. Nakajima, H. Varieties Associated with Quivers. In Proceedings of the Representation Theory of Algebras and Related Topics, CMS Conference Proceedings, Geiranger, Norway, 4–10 August 1996; Volume 19, pp. 139–157.
46. Munkres, J. *Topology*; Featured Titles for Topology; Prentice Hall, Incorporated: Hoboken, NJ, USA, 2000.
47. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
48. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In *International Conference on Neural Information Processing Systems—Volume 2*; MIT Press: Cambridge, MA, USA, 2014; pp. 2672–2680.
49. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic Policy Gradient Algorithms. In Proceedings of the International Conference on Machine Learning–Volume 32, Beijing China, 21–26 June 2014; pp. I-387–I-395.
50. Hopfield, J., Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *Neurocomputing: Foundations of Research*; MIT Press: Cambridge, MA, USA, 1988; pp. 457–464.
51. Baxter, J., Theoretical Models of Learning to Learn. In *Learning to Learn*; Kluwer Academic Publishers: Drive Norwell, MA, USA, 1998; pp. 71–94.
52. Hartshorne, R. *Algebraic Geometry*; Graduate Texts in Mathematics; Springer: New York, NY, USA, 1977; Volume 52, pp. XVI, 496.
53. Cox, D.; Little, J.; Schenck, H. *Toric Varieties*; Graduate studies in mathematics; American Mathematical Society Providence: Providence, RI, USA, 2011; Volume 124.
54. Hille, L. Toric Quiver Varienties. *Can. Math. Soc. Conf. Proc.* **1998**, *24*, 311–325.
55. Domokos, M.; Joó, D. On the Equations and Classification of Toric Quiver Varieties. *Proc. R. Soc. Edinburgh Sect. A Math.* **2016**, *146*, 265–295. [CrossRef]
56. De la Peña, J.A.; Gabriel, P. On Algebras, Wild and Tame. In *Duration and Change*; Artin M., Remmert R., Kraft, H., Eds.; Springer: Berlin/Heidelberg, Germany, 1994.
57. Zimmermann, A. *Representation Theory: A Homological Algebra Point of View*; Algebra and Applications; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; Volume 19, pp. xx, 707.