PARAMETERIZATION OF HYPERCOMPLEX MULTIPLICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent works have demonstrated reasonable success of representation learning in hypercomplex space. Specifically, the Hamilton product (4D hypercomplex multiplication) enables learning effective representations while saving up to 75% parameters. However, one key caveat is that hypercomplex space only exists at very few predefined dimensions. This restricts the flexibility of models that leverage hypercomplex multiplications. To this end, we propose parameterizing hypercomplex multiplications, allowing models to learn multiplication rules from data regardless of whether such rules are predefined. As a result, our method not only subsumes the Hamilton product, but also learns to operate on any arbitrary nD hypercomplex space, providing more architectural flexibility. Experiments of applications to LSTM and Transformer on natural language inference, machine translation, text style transfer, and subject verb agreement demonstrate architectural flexibility and effectiveness of the proposed approach.

1 INTRODUCTION

A Quaternion is a 4D hypercomplex number with one real component and three imaginary components. The Hamilton product is the hypercomplex multiplication of two Quaternions. Recent works in Quaternion space and Hamilton products have demonstrated reasonable success (Parcollet et al., 2018b; 2019; Tay et al., 2019). The Hamilton product enjoys a parameter saving with $1/4$ learnable parameters as compared with the real-valued matrix multiplication. It also enables effective representation learning by modeling interactions between real and imaginary components.

One of the attractive properties of Quaternion models (Parcollet et al., 2018b; 2019; Tay et al., 2019) is its high applicability and universal usefulness to one of the most ubiquitous layers in deep learning, i.e., the fully-connected (or feed-forward) layer. This transformation layer is one of the most dominant component in existing deep learning literature (Goodfellow et al., 2016; Zhang et al., 2020). Its pervasiveness cannot be understated, given its centrality to many core building blocks in neural network research. Given widespread adoptions of fully-connected layers, e.g., within LSTM networks (Hochreiter & Schmidhuber, 1997) and Transformer models (Vaswani et al., 2017), having flexibility to balance between parameter savings and effectiveness could be extremely useful to many real-world applications.

Unfortunately, hypercomplex space only exists at 4D (Quaternions), 8D (Octonions), and 16D (Sedonions), which generalizes the 2D complex space (Rishiyur, 2006). Moreover, custom operators are required at each hypercomplex dimensionality. For instance, the Hamilton product is the hypercomplex multiplication in 4D hypercomplex space. Thus, no operator in such predefined hypercomplex space is suitable for applications that prefer reducing parameters to $1/n$, where $n \neq 4, 8, 16$.

In view of the architectural limitation due to the very few choices of existing hypercomplex space, we propose parameterization of hypercomplex multiplications, i.e., learning the real and imaginary component interactions from data in a differentiable fashion. Essentially, our method can operate on an arbitrary nD hypercomplex space, aside from subsuming those predefined hypercomplex multiplication rules, facilitating using up to $1/n$ learnable parameters while maintaining expressiveness. In practice, the hyperparameter n can be flexibly specified or tuned by users based on applications.

Concretely, our prime contribution is a new module that parameterizes and generalizes the hypercomplex multiplication by learning the real and imaginary component interactions, i.e., multiplica-

tion rules, from data. Our method, which we call the parameterized hypercomplex multiplication layer, is characterized by a sum of Kronecker products that generalize the vector outer products to higher dimensions in real space. To demonstrate applicability, we equip two well-established models (LSTM and Transformer) with our proposed method. We conduct extensive experiments on different tasks, i.e., natural language inference for LSTM and machine translation for Transformer. Additionally, we perform further experiments on text style transfer and subject verb agreement tasks. All in all, our method has demonstrated architectural flexibility through different experimental settings, where it generally can use a fraction of the learnable parameters with minimal degradation or slight improvement in performance.

The overall contributions of this work are summarized as follows:

- We propose a new parameterization of hypercomplex multiplications: the parameterized hypercomplex multiplication (PHM) layers. The key idea behind PHM layers is to learn the interactions between real and imaginary components, i.e., multiplication rules, from data using a sum of Kronecker products.
- We demonstrate the applicability of the PHM layers by leveraging them in two dominant neural architectures: LSTM and Transformer.
- We empirically show architectural flexibility and effectiveness of PHM layers by conducting extensive experiments on five natural language inference tasks, seven machine translation datasets, together with text style transfer and subject verb agreement tasks.

2 Background on Quaternions and Hamilton Products

We begin by introducing the background for the rest of the paper. Concretely, we describe Quaternion algebra along with Hamilton products, which is at the heart of our proposed approach.

Quaternion A Quaternion $Q \in \mathbb{H}$ is a hypercomplex number with one real component and three imaginary components as follows:

$$Q = Q_r + Q_x\mathbf{i} + Q_y\mathbf{j} + Q_z\mathbf{k}, \tag{2.1}$$

whereby $\mathbf{ijk} = \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$. In equation 2.1, noncommutative multiplication rules hold: $\mathbf{ij} = \mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{ki} = \mathbf{j}, \mathbf{ji} = -\mathbf{k}, \mathbf{kj} = -\mathbf{i}, \mathbf{ik} = -\mathbf{j}$. Here, Q_r is the real component, Q_x, Q_y, Q_z are real numbers that represent the imaginary components of the Quaternion Q.

Addition The addition of two Quaternions is defined as

$$Q + P = Q_r + P_r + (Q_x + P_x)\mathbf{i} + (Q_y + P_y)\mathbf{j} + (Q_z + P_z)\mathbf{k},$$

where Q and P with subscripts denote the real and imaginary components of Quaternions Q and P.

Scalar Multiplication Any scalar α multiplies across all the components:

$$\alpha Q = \alpha Q_r + \alpha Q_x\mathbf{i} + \alpha Q_y\mathbf{j} + \alpha Q_z\mathbf{k}.$$

Hamilton Product The Hamilton product, which represents the multiplication of two Quaternions Q and P, is defined as

$$\begin{aligned} Q \otimes P = &(Q_rP_r - Q_xP_x - Q_yP_y - Q_zP_z) + (Q_xP_r + Q_rP_x - Q_zP_y + Q_yP_z)\,\mathbf{i} \\ &+ (Q_yP_r + Q_zP_x + Q_rP_y - Q_xP_z)\,\mathbf{j} + (Q_zP_r - Q_yP_x + Q_xP_y + Q_rP_z)\,\mathbf{k}. \end{aligned} \tag{2.2}$$

The multiplication rule in equation 2.2 forges interactions between real and imaginary components of Q and P. The benefits of Hamilton products have been demonstrated in recent works where the matrix multiplication in fully-connected layers is replaced with the Hamilton product: this reduces 75% parameters with comparable performance (Parcollet et al., 2018b; 2019; Tay et al., 2019).

3 Parameterization of Hypercomplex Multiplications

The following introduces our proposed parameterized hypercomplex multiplication layer and elaborates on how it parameterizes and generalizes multiplications in hypercomplex space, such as subsuming the multiplication rules of Hamilton products in equation 2.2.

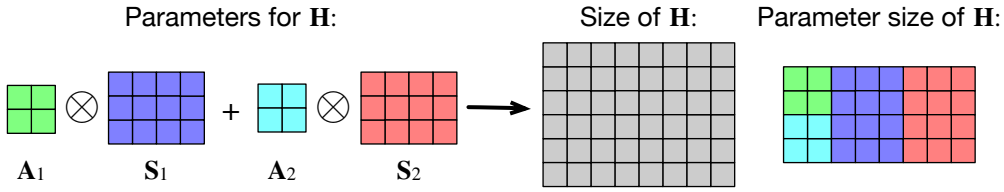

Figure 1: Illustration of the PHM layer. It uses a sum of Kronecker products of matrices $\mathbf{A}_i$ and $\mathbf{S}_i$ ($i = 1, 2$) to construct $\mathbf{H}$ in equation 3.2 (here $n = 2, k = 6, d = 8$). Best viewed in color.

3.1 Fully-Connected (FC) Layers

Before we delve into our proposed method, recall the fully-connected (FC) layer that transforms an input $\mathbf{x} \in \mathbb{R}^d$ into an output $\mathbf{y} \in \mathbb{R}^k$ by

$$\mathbf{y} = \text{FC}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \boldsymbol{b}, \tag{3.1}$$

where the weight matrix of parameters $\mathbf{W} \in \mathbb{R}^{k \times d}$ and the bias vector of parameters $\boldsymbol{b} \in \mathbb{R}^k$. The FC layer in equation 3.1 is fundamental to many modern and traditional neural network architectures. Note that the degree of freedom for the weight parameters $\mathbf{W}$ in equation 3.1 is kd. Since $\mathbf{W}$ dominates parameterization, the parameter size of the FC layer in equation 3.1 is $\mathcal{O}(kd)$.

3.2 Parameterized Hypercomplex Multiplication (PHM) Layers

We propose the parameterized hypercomplex multiplication (PHM) layer that transforms an input $\mathbf{x}$ into an output $\mathbf{y}$ by

$$\mathbf{y} = \text{PHM}(\mathbf{x}) = \mathbf{H}\mathbf{x} + \boldsymbol{b}, \tag{3.2}$$

where the same notation from equation 3.1 is used but the replaced parameter $\mathbf{H} \in \mathbb{R}^{k \times d}$ is constructed by a sum of Kronecker products. For context, the Kronecker product is a generalization of the vector outer product to higher dimensions in real space. For any matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{p \times q}$, the Kronecker product $\mathbf{X} \otimes \mathbf{Y}$ is a block matrix:

$$\mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{11}\mathbf{Y} & \dots & x_{1n}\mathbf{Y} \\ \vdots & \ddots & \vdots \\ x_{m1}\mathbf{Y} & \dots & x_{mn}\mathbf{Y} \end{bmatrix} \in \mathbb{R}^{mp \times nq},$$

where x_{ij} is the element of $\mathbf{X}$ at its ith row and j^{th} column. Note that the symbol $\otimes$ between two matrices is the Kronecker product while the same symbol between two Quaternions means the Hamilton product.

Now let us revisit equation 3.2 to explain $\mathbf{H}$. Suppose that both k and d are divisible by a user-defined hyperparameter $n \in \mathbb{Z}_{>0}$. For $i = 1, \dots, n$, denote by each parameter matrix $\mathbf{A}_i \in \mathbb{R}^{n \times n}$ and $\mathbf{S}_i \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$. The parameter $\mathbf{H}$ in equation 3.2 is a sum of n Kronecker products:

$$\mathbf{H} = \sum_{i=1}^{n} \mathbf{A}_i \otimes \mathbf{S}_i. \tag{3.3}$$

As illustrated in Figure 1, it is the parameter matrices $\mathbf{A}_i$ and $\mathbf{S}_i$ ($i = 1, \dots, n$) that determine the degree of freedom for $\mathbf{H}$, which is $kd/n + n^3$. Since $\mathbf{H}$ dominates parameterization, the parameter size of the PHM in equation 3.2 is $\mathcal{O}(kd/n)$, where $kd \gtrapprox n^4$ is assumed: this condition is mild for real-world problems, such as in our experiments (e.g., $d = 512$, $k = 2048$, $n = 2, 4, 8, 16$). Thus, for the same input and output sizes, the parameter size of a PHM layer is approximately $1/n$ of that of an FC layer under mild assumptions.

The benefit of parameterization reduction of PHM layers is due to reusing elements of both parameter matrices $\mathbf{A}_i$ and $\mathbf{S}_i$ in the Kronecker product. As an alternative perspective, we can equivalently

reconstruct $\mathbf{H}$ in equation 3.3 by reusing parameter matrices in real-valued matrix multiplications, followed by more operations. Due to limited space, this more complicated perspective is offered in the supplementary materials.

As highlighted in our contributions, our goal is to parameterize hypercomplex multiplications to offer architectural flexibility rather than to compress the matrix $\mathbf{H}$ in equation 3.2. Thus, though simply setting $\mathbf{H} = \mathbf{A}_1 \otimes \mathbf{S}_1$ can further save parameters, it does not generalize hypercomplex multiplications hence is out of scope. In the following, we show how the proposed PHM layer subsumes and generalizes both hypercomplex multiplications and real-valued matrix multiplications.

3.3 Subsuming Hypercomplex Multiplications

First, we explore how the PHM layer connects to the hypercomplex multiplication. For the sake of illustration, let us take the Hamilton product of two Quaternions Q and P in equation 2.2 as an example, which can be rewritten as

$$\begin{bmatrix} Q_r & -Q_x & -Q_y & -Q_z \\ Q_x & Q_r & -Q_z & Q_y \\ Q_y & Q_z & Q_r & -Q_x \\ Q_z & -Q_y & Q_x & Q_r \end{bmatrix} \begin{bmatrix} P_r \\ P_x \\ P_y \\ P_z \end{bmatrix}, \tag{3.4}$$

where the 4 output elements are the real values for the Quaternion unit basis $[1, \mathbf{i}, \mathbf{j}, \mathbf{k}]^\top$. Note that for models leveraging Hamilton products of Quaternions (Parcollet et al., 2018b; 2019; Tay et al., 2019), the components Q_r, Q_x, Q_y, Q_z of equation 3.4 are learnable parameters while the components P_r, P_x, P_y, P_z are the layer inputs. In practice, such a layer usually has more than 4 inputs ($d > 4$). To apply the Hamilton product, all the inputs are evenly split into 4 segments (P_r, P_x, P_y, P_z) of the right input vector of equation 3.4. Then each component in the left matrix of equation 3.4 can be a block matrix (i) where all the elements take the same value; (ii) whose shape is aligned with the input length d and the output length k of the layer. It is noteworthy that the left 4×4 matrix of equation 3.4 can be rewritten as a sum of 4 Kronecker products:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{A}_1} \otimes \underbrace{\begin{bmatrix} Q_r \end{bmatrix}}_{\mathbf{S}_1} + \underbrace{\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{A}_2} \otimes \underbrace{\begin{bmatrix} Q_x \end{bmatrix}}_{\mathbf{S}_2} + \underbrace{\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}}_{\mathbf{A}_3} \otimes \underbrace{\begin{bmatrix} Q_y \end{bmatrix}}_{\mathbf{S}_3} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}_4} \otimes \underbrace{\begin{bmatrix} Q_z \end{bmatrix}}_{\mathbf{S}_4}. \tag{3.5}$$

According to equation 3.5, when $n = 4$, the PHM layer can be learned to express the Hamilton product of Quaternions. Specifically, matrices $\mathbf{A}_1, \ldots, \mathbf{A}_4$ in equation 3.3 parameterize the four matrices composed of $-1, 0, 1$ in equation 3.5 that reflect interactions between real and imaginary components of Quaternions, which are the rule of Hamilton products. The single-element "matrices" $\mathbf{S}_1, \ldots, \mathbf{S}_4$ in equation 3.3 are equal to the learnable components Q_r, Q_x, Q_y, Q_z in equation 3.4. Likewise, hypercomplex multiplications of Octonions or Sedenions can also be learned by the PHM layer when n is set to 8 or 16.

3.4 Subsuming Real-Valued Matrix Multiplications

Next, we show how the PHM layer subsumes the matrix multiplication in real space. In other words, the PHM layer is a generalization of the FC layer via the hyperparameter n. To explain, referring to equation 3.2, when $n = 1$, $\mathbf{H} = \mathbf{A}_1 \otimes \mathbf{S}_1 = a\mathbf{S}_1$, where the scalar a is the single element of the 1×1 matrix $\mathbf{A}_1$ and $\mathbf{S}_1 \in \mathbb{R}^{k \times d}$. Since learning a and $\mathbf{S}_1$ separately is equivalent to learning their multiplication jointly, scalar a can be dropped, which is learning the single weight matrix in an FC layer. Therefore, a PHM layer is degenerated to an FC layer when $n = 1$.

3.5 Generalizing Hypercomplex Multiplications

Though parameter reusing by component-wise partitioning in Quaternion space has demonstrated success (Parcollet et al., 2018b; Zhu et al., 2018; Parcollet et al., 2019; Tay et al., 2019), one key problem is that hypercomplex space only exists at very few predefined dimensionalities, such as 4D

(Quaternions), 8D (Octonions), and 16D (Sedonions). Within the context of hypercomplex space, specialized multiplication rules, such as the Hamilton product, have to be devised and encoded in the network as a fixed inductive bias. As described in Section 1, the very few choices over existing hypercomplex space restricts the flexibility of networks that leverage hypercomplex multiplication.

In sharp contrast to relying on predefined mathematical rules over limited dimensionality choices, the PHM layer treats the dimensionality n (number of Kronecker products) as a tunable hyperparameter and learns such specialized multiplication rules from data, as manifested in the parameterized matrices $\mathbf{A}_i$ $(i = 1, \ldots, n)$ in equation 3.3. On one hand, the PHM layer can express hypercomplex multiplications when $\mathbf{A}_i$ are set to reflect those predefined multiplication rules in hypercomplex space. On the other hand, the PHM layer can be seen as a trainable and parameterized form of nD hypercomplex multiplications, where n can be values other than 4, 8, or 16. Thus, the PHM layer generalizes multiplications in hypercomplex space. Since n can be 1, the PHM layer also offers a neat way to bridging multiplication between both real space and hypercomplex space.

4 Neural Models with PHM Layers

To demonstrate the applicability of the PHM layers, we develop PHM-LSTM and PHM-Transformer by equipping two popular neural network models, LSTM and Transformer, with PHM layers.

4.1 PHM-LSTM

Recurrent neural networks such as LSTM (Hochreiter & Schmidhuber, 1997) are gated recurrent networks where the gating functions are parameterized by linear transformations. We introduce PHM-LSTM, which replaces such linear transformations in LSTM with PHM layers:

$$\begin{aligned}
\mathbf{y}_t &= \text{PHM}\left(\mathbf{x}_t\right) + \text{PHM}\left(\mathbf{h}_{t-1}\right) + \boldsymbol{b} \\
\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t, \mathbf{x}'_t &= \phi(\mathbf{y}_t) \\
\mathbf{c}_t &= \sigma_s(\mathbf{f}_t)\, \mathbf{c}_{t-1} + \sigma_s(\mathbf{i}_t)\, \sigma_t(\mathbf{x}'_t) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \mathbf{c}_t,
\end{aligned}$$

where σ_s is the sigmoid activation function, σ_t is the tanh activation function, $\phi : \mathbb{R}^{1\times d} \rightarrow \mathbb{R}^{4\times \frac{d}{4}}$ is a four-way split on the last dimension, and $\mathbf{c}_t, \mathbf{h}_t$ are the cell state and the hidden state of the PHM-LSTM unit at any time step t.

4.2 PHM-Transformer

Transformer is a stacked neural network architecture that aggressively exploits linear transformations (Vaswani et al., 2017). Each self-attention layer comprises of $\mathbf{Q}$ (query), $\mathbf{K}$ (key), $\mathbf{V}$ (value) linear transformations, along with multiple heads. Each Transformer block also has a position-wise feed-forward network composed of two FC layers. Since a large majority of the Transformer parameters stem from linear transformations or FC layers, we introduce PHM-Transformer to replace all the linear transformations or FC layers with PHM layers. The single-head self-attention module is rewritten as:

$$\begin{aligned}
\mathbf{Q}, \mathbf{K}, \mathbf{V} &= \Phi(\text{PHM}(\mathbf{X})) \\
\mathbf{A} &= \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V},
\end{aligned}$$

where d_k is the key dimension, $\Phi : \mathbb{R}^{1\times d} \rightarrow \mathbb{R}^{3\times \frac{d}{3}}$ is a three-way split on the last dimension, $\mathbf{X}$ is the input sequence, and $\mathbf{A}$ is the self-attentive representation. For multi-head attention, using PHM layers also enables weight sharing not only among the linear transformations of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ but also among the linear transformation of multiple heads:

$$\mathbf{X} = \text{PHM}([\mathbf{H}_1; \ldots; \mathbf{H}_{N_h}]),$$

where N_h is the number of heads and (;) is the column-wise concatenation. Finally, the position-wise feed-forward network is now defined as

$$\mathbf{Y} = \text{PHM}(\text{ReLU}(\text{PHM}(\mathbf{X}))),$$

which transforms $\mathbf{X}$ with two PHM layers.

Table 1: Experimental results of natural language inference (accuracy) on five different datasets. PHM-LSTM reduces the parameters of the standard LSTM model and improves or partially matches performance on four out of five datasets.

Model	#Params	MNLI	QNLI	SNLI	DNLI	SciTail
LSTM	721K	**71.82** / 71.89	84.44	84.18	85.16	74.36
Quaternion LSTM	180K (-75.0%)	71.57 / **72.19**	**84.73**	84.21	86.45	75.58
PHM-LSTM ($n = 2$)	361K (-49.9%)	**71.82** / 72.08	84.39	84.38	85.77	77.47
PHM-LSTM ($n = 5$)	146K (-79.7%)	71.80 / 71.77	83.87	**84.58**	**86.47**	74.64
PHM-LSTM ($n = 10$)	81K (-88.7%)	71.59 / 71.59	84.25	84.40	86.21	**77.84**

5 EXPERIMENTS

We reiterate that our contribution is parameterization of hypercomplex multiplications rather than a model compression technique without requirements of generalizing hypercomplex multiplications. In the field of representation learning using hypercomplex multiplications, Quaternion convolutional neural networks (Zhu et al., 2018), Quaternion recurrent neural networks (Parcollet et al., 2018a), and Quaternion Transformer (Tay et al., 2019) have all compared themselves with only real-valued counterparts. Therefore, to support our contribution and be consistent with the rest of the literature, we evaluate PHM-LSTM and PHM-Transformer that are equipped with PHM layers, and compare them with Quaternion LSTM, Quaternion Transformer, or real-valued LSTM or Transformer. Both Quaternion LSTM and Quaternion Transformer replace linear transformations with Hamilton products of Quaternions.

To demonstrate the architectural and effectiveness, we evaluate different settings of PHM-LSTM and PHM-Transformer to show that allowing for flexible choices of the hyperparameter n in the PHM layer may lead to more effective performance. Details of the setup for the experiments are provided in the supplementary materials.

5.1 NATURAL LANGUAGE INFERENCE

The task of natural language inference is to determine the logical relationship between two text sequences (MacCartney, 2009). It is a fundamental task pertaining to language understanding. To this end, they serve as a suitable benchmark for evaluating recurrent models.

We run experiments on five datasets: (i) MultiNLI (Williams et al., 2017), (ii) QNLI (Quora) (Wang et al., 2017), (iii) SNLI (Bowman et al., 2015), (iv) Dialogue NLI (Welleck et al., 2018), and (v) SciTail (Science Entailment) (Khot et al., 2018). Table 1 reports the results on all these datasets. All in all, such results show that the PHM layer can not only reduce the parameters but also improve performance with flexible choices of n (four out of five datasets show reasonable improvement or partially match). The only exception is on the QNLI dataset, where the performance drop is marginal ($< 1\%$). This is still decent considering the parameter saving: the parameterization cost of PHM-LSTM is in the order of $\mathcal{O}(1/n)$ of that of the standard LSTM, where settings of $n = 5$ and $n = 10$ do not take values of power of 2. As detailed in the supplementary materials, since we use the 300D GloVe (Pennington et al., 2014) embeddings to represent input tokens, we choose multiples of 5 instead of 4 for ease of divisibility. It is also noteworthy that on the SNLI, Dialogue NLI, and SciTail datasets, all the PHM-LSTM variants outperform the standard LSTM model. We think that the element reusing properties of the Kronecker product operation, in addition to learning to share such reused components amongst recurrent gating functions, may contribute to both effective and efficient representations.

5.2 MACHINE TRANSLATION

Machine translation is concerned with translating between source-target language pairs. To this end, sequence transduction models are central to this problem domain. In this experiment, the key goal is to compare PHM-Transformer against the standard and Quaternion Transformer models.

Table 2: Experimental results of machine translation (BLEU) on seven different datasets. Symbol † represents re-scaling the parameters with a factor of 2 by doubling the hidden size. PHM-Transformer does not lose much performance despite enjoying parameter savings. Re-scaling can lead to improvement in performance.

Model	#Params	En-Vi	En-Id	De-En	Ro-En	En-Et	En-Mk	En-Ro
Transformer (Tm)	44M	28.43	47.40	**36.68**	**34.60**	14.17	13.96	**22.79**
Quaternion Tm	11M (-75.0%)	28.00	42.22	32.83	30.53	13.10	13.67	18.50
PHM-Tm $n = 2$	22M (-50.0%)	29.25	46.32	35.52	33.40	**14.98**	13.60	21.73
PHM-Tm $n = 4$	11M (-75.0%)	29.13	44.13	35.53	32.74	14.11	13.01	21.19
PHM-Tm $n = 8$	5.5M (-87.5%)	29.34	40.81	34.16	31.88	13.08	12.95	21.66
PHM-Tm $n = 16$	2.9M (-93.4%)	29.04	33.48	33.89	31.53	12.15	11.97	19.63
PHM-Tm† $n = 2$	44M	**29.54**	**49.05**	34.32	33.88	14.05	**14.41**	22.18
PHM-Tm† $n = 4$	22M (-50.0%)	29.17	46.24	34.86	33.80	14.43	13.78	21.91
PHM-Tm† $n = 8$	11M (-75.0%)	29.47	43.49	34.71	32.59	13.75	13.78	21.43

We run experiments on seven datasets: (i) IWSLT'15 English-Vietnamese (En-Vi), (ii) IWSLT'17 English-Indonesian (En-Id), (iii) IWSLT'14 German-English (De-En), (iv) IWSLT'14 Romanian-English (Ro-En), (v) WMT'18 English-Estonian (En-Et), (vi) Setimes English-Macedonian (En-Mk), and (vii) WMT'18 English-Romanian (En-Ro).

Table 2 reports our results of the machine translation tasks. Overall, these empirical results with different settings demonstrate architectural flexibility and effectiveness of the hypercomplex multiplication parameterization. First and foremost, across six out of seven benchmarks, PHM-Transformer at $n = 4$ makes reasonable gains over Quaternion Transformer, signifying that parameterization of hypercomplex multiplications by learning from data can be more effective than predefining Hamilton product rules mathematically. Second, though increasing n leads to more parameter savings, we observe that increasing n all the way to 16 does not cause significant degradation in performance on datasets such as En-Vi. Third, for most datasets, even with significant parameter savings, we find that the decrease in the BLEU score is mostly manageable ($\approx$ 1–3 BLEU points). However, we also note a rare occurrence where $n = 16$ results in a significant decrease in the BLEU score, such as on the En-Id dataset. Fourth, on several datasets, the PHM-Transformer model improves the performance of the standard Transformer model. For example, on datasets such as En-Vi and En-Et, the PHM-Transformer model enjoys a performance boost of about 0.8 BLEU point with $n = 2$. Finally, by re-scaling with a factor of 2 (doubling the hidden size), we are able to improve the performance on three datasets: En-Vi, En-Id, and En-Mk.

Table 3: Training time (seconds per 100 steps) and inference time (seconds to decode test sets) with beam size of 4 and length penalty of 0.6 on the IWSLT'14 German-English dataset.

Model	Transformer (Tm)	Quaternion Tm	PHM-Tm ($n = 4$)	PHM-Tm ($n = 8$)
Training time	**7.79**	8.31	8.09	7.89
Inference time	341	297	303	**287**

Table 3 reports the training and inference time for Transformer variants. We observe that PHM-Transformer with $n = 8$ has the fastest inference speed amongst all the variants, primarily due to a significant reduction of parameters. All in all, the training speed is also approximately comparable. This ascertains that the PHM layer does not increase much computational cost in practice.

5.3 Text Style Transfer

We continue to experiment with sequence transduction for text style transfer. The goal of this task is to convert text of a certain style to another style. We use the Modern→Shakespeare corpus[1] in the experiments. Table 4 reports the results on this text style transfer task. We observe that the best

[1] https://github.com/tlatkowski/st

performance is achieved with PHM-Transformer ($n = 4$). Notably, all except the $n = 16$ variant increases or matches the performance of the standard Transformer model. This ascertains architectural flexibility and effectiveness of the proposed PHM layer. This not only enables parameter savings but also improves the performance of Transformer.

Table 4: Experimental results of text style transfer. PHM-Transformer may reduce the parameters of the standard Transformer model and improve performance.

Model	#Params	BLEU
Transformer (Tm)	44M	11.65
PHM-Tm ($n = 2$)	22M (-50.0%)	12.20
PHM-Tm ($n = 4$)	11M (-75.0%)	**12.42**
PHM-Tm ($n = 8$)	5.5M (-87.5%)	11.66
PHM-Tm ($n = 16$)	2.9M (-93.4%)	10.76

Table 5: Experimental results of subject verb agreement. PHM-Transformer may reduce the parameters of the standard Transformer model and improve performance.

Model	#Params	Acc
Transformer (Tm)	400K	94.80
Quaternion Tm	100K	94.70
PHM-Tm ($n = 2$)	200K (-50.0%)	95.14
PHM-Tm ($n = 4$)	101K (-74.8%)	95.05
PHM-Tm ($n = 8$)	56K (-86.0%)	**95.62**

5.4 Subject Verb Agreement

We conduct additional experiments on the subject-verb agreement task (Linzen et al., 2016). The task predicts if the sentence, e.g., *'The keys to the cabinet _____ .'* is followed by a plural or a singular. The used dataset can be found online (Linzen et al., 2016). Table 5 reports the results on the subject-verb agreement task. Results are promising, demonstrating that all variants with PHM layers outperform the standard and Quaternion Transformer models. The best performance peaks at $n = 8$, despite a parameter saving to up to $1/8$.

6 Related Work

While neural networks have been a well-established line of research, progress on hypercomplex representations for deep learning is still in its infancy and most works on this topic are new (Gaudet & Maida, 2017; Parcollet et al., 2018a;b; Zhu et al., 2018; Tay et al., 2019). The hypercomplex Hamilton product provides a greater extent of expressiveness, similar to the complex multiplication, albeit with a 4-fold increase in interactions between real and imaginary components. In the case of Quaternion representations, due to parameter savings in the Hamilton product, models also enjoy a 75% reduction in the parameter size (Parcollet et al., 2018a; Tay et al., 2019). A striking caveat is that all Quaternions are fundamentally limited to 4D hypercomplex space, which restricts architectural flexibility. The other options would be to scale to Octonion (8D) or Sedonion (16D) space, given the predefined multiplication rules in such space. To the best of our knowledge, there is no work that attempts to generalize arbitrary nD hypercomplex multiplications to allow for architectural flexibility, where n can be specified or tuned by users.

Our work can also be interpreted as a form of soft parameter sharing, albeit learned from data. Quaternion networks (Zhu et al., 2018; Parcollet et al., 2018b; 2019) are known to possess weight sharing properties via the Hamilton product operation and have demonstrated reasonable success despite having fewer parameters. To the best of our knowledge, there has been no work that attempts to parameterize the hypercomplex Hamilton product for neural networks, i.e., enabling end-to-end learning of real and imaginary component interactions from data.

7 Conclusion

We proposed parameterized hypercomplex multiplication (PHM) layers that learn and generalize hypercomplex multiplications. PHM layers are highly modular and applicable to dominant models such as LSTM and Transformer. We evaluated these models equipped by PHM layers on comprehensive tasks to demonstrate architectural flexibility and effectiveness of the hypercomplex multiplication parameterization.

REFERENCES

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Chase Gaudet and Anthony Maida. Deep quaternion networks. *arXiv preprint arXiv:1712.04604*, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Tushar Khot, Ashish Sabharwal, and Peter Clark. Scitail: A textual entailment dataset from science question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016.

Bill MacCartney. *Natural language inference*. Citeseer, 2009.

Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linarès, Chiheb Trabelsi, Renato De Mori, and Yoshua Bengio. Quaternion recurrent neural networks. In *International Conference on Learning Representations*, 2018a.

Titouan Parcollet, Ying Zhang, Mohamed Morchid, Chiheb Trabelsi, Georges Linarès, Renato De Mori, and Yoshua Bengio. Quaternion convolutional neural networks for end-to-end automatic speech recognition. *arXiv preprint arXiv:1806.07789*, 2018b.

Titouan Parcollet, Mohamed Morchid, and Georges Linarès. Quaternion convolutional neural networks for heterogeneous image processing. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8514–8518. IEEE, 2019.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Adityan Rishiyur. Neural networks with complex and quaternion inputs. *arXiv preprint cs/0607090*, 2006.

Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv preprint arXiv:1906.04393*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.

Sean Welleck, Jason Weston, Arthur Szlam, and Kyunghyun Cho. Dialogue natural language inference. *arXiv preprint arXiv:1811.00671*, 2018.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. `https://d2l.ai`.

Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. Quaternion convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 631–647, 2018.