

## A ADDITIONAL ALGEBRA INFORMATION

### A.1 $M_2(\mathbb{R})$ MULTIPLICATION TABLE

Each tuple  $(t_a, t_b, t_c, t_d)$  represents a  $2 \times 2$  real matrix.

$M_2(\mathbb{R})$	$a$	$b$	$c$	$d$
$a$	$a$	$b$	$0$	$0$
$b$	$0$	$0$	$a$	$b$
$c$	$c$	$d$	$0$	$0$
$d$	$0$	$0$	$c$	$d$

### A.2 $M_2(\mathbb{C})$ MULTIPLICATION TABLE

Each tuple  $(t_a, t_b, t_c, t_d, t_e, t_f, t_g, t_h)$  represents the  $2 \times 2$  complex matrix:  $\begin{bmatrix} t_a + t_b i & t_c + t_d i \\ t_e + t_f i & t_g + t_h i \end{bmatrix}$

$M_2(\mathbb{C})$	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$
$a$	$a$	$b$	$c$	$d$	$0$	$0$	$0$	$0$
$b$	$b$	$-a$	$d$	$-c$	$0$	$0$	$0$	$0$
$c$	$0$	$0$	$0$	$0$	$a$	$b$	$c$	$d$
$d$	$0$	$0$	$0$	$0$	$b$	$-a$	$d$	$-c$
$e$	$e$	$f$	$g$	$h$	$0$	$0$	$0$	$0$
$f$	$f$	$-e$	$h$	$-g$	$0$	$0$	$0$	$0$
$g$	$0$	$0$	$0$	$0$	$e$	$f$	$g$	$h$
$h$	$0$	$0$	$0$	$0$	$f$	$-e$	$h$	$-g$

### A.3 DUAL NUMBER MULTIPLICATION TABLE

Each tuple  $(t_a, t_b)$  represents the dual number  $(t_a + t_b \epsilon)$ .

Dual Number	$a$	$b$
$a$	$a$	$b$
$b$	$b$	$0$

### A.4 CROSS PRODUCT MULTIPLICATION TABLE

Multiplicated uses the cross product between length-3 tuples  $(t_a, t_b, t_c)$ .

Cross Product	$a$	$b$	$c$
$a$	$0$	$c$	$-b$
$b$	$-c$	$0$	$a$
$c$	$b$	$-a$	$0$

### A.5 LINEAR LAYER EXAMPLE

We give a concrete example of replacing a real linear layer with  $M_2(\mathbb{R})$ -linear layer such that the activation memory is kept identical. Intuitively, this can be thought of as reshaping the  $\mathbb{R}^d$  input activations to have shape  $M_2(\mathbb{R})^{d/4}$  that is processed by a  $f_M : M_2(\mathbb{R})^{d/4} \rightarrow M_2(\mathbb{R})^{d/4}$  linear layer resulting in output activations – when flattened – with shape  $\mathbb{R}^d$ . Each such linear layer  $f_M$  requires  $\frac{1}{4}$  of the parameters and  $\frac{1}{2}$  of the FLOPS compared to a real  $\mathbb{R}^d \rightarrow \mathbb{R}^d$  linear layer counterpart.

## B ALGEBRANET CHOICES: ACTIVATIONS, INITIALIZATIONS, ETC

### B.1 TUPLE-WISE NONLINEARITY

We consider equations of the form:

$$\mathbf{t} \leftarrow f(g(\mathbf{t})) * \mathbf{t} \quad (1)$$

We found that if  $g$  is the tuple mean, and  $f$  is  $H$  the Heaviside function, top-1 performance dropped on an  $M_2(\mathbb{R})$  ResNet-50 AlgebraNet by 2.97%. While this drop is significant, the resulting activation sparsity might make it a desirable tradeoff in some circumstances. Other methods, such as setting  $g$  to be the determinant resulted in greater than a 10% drop in performance.

## B.2 INITIALIZATION

For a ResNet-50  $\mathbb{H}$ -AlgebraNet with the standard number of channels divided by 4, we find a top-1 performance of  $74.0 \pm 0.14$  using standard initialization and  $74.1 \pm 0.15$  using initialization from Gaudet and Maida (2018). These experiments are done using standard batch normalization instead of the more expensive whitening procedure.

## B.3 CONVERSION TO REALS

For all considered algebras, the norm of the tuple is mathematically given by  $\sqrt{\sum_i t_i^2}$ . It is possible that the optimal choice for converting to the reals would be different in models with very large final layers, such as word based language modeling – which we do not consider.

# C ALGEBRA NET PRUNING

## C.1 ALTERNATIVE TUPLE PRUNING OF $M_2(\mathbb{R})$

For  $M_2(\mathbb{R})$ , we consider a variety of alternative pruning methods to remove entire tuples, based on the two eigenvalues,  $\lambda_1$  and  $\lambda_2$  and singular values,  $\sigma_1, \sigma_2$ . Specifically, because our matrices are square but not symmetric, the Frobenius norm is defined based on the singular values which correspond to the squared eigenvalues of  $AA^T$ , if  $A$  is the matrix in question.

- Frobenius Norm:  $(\sigma_1^2 + \sigma_2^2)^{1/2}$
- Determinant:  $\lambda_1 \lambda_2$
- Smallest Eigenvalue  $\min(|\lambda_1|, |\lambda_2|)$
- Largest Eigenvalue  $\max(|\lambda_1|, |\lambda_2|)$

In all cases, we remove tuples with the minimum magnitude of one of those options.

Sparsity	det	min	max
50	-0.27	-0.76	-0.02
70	-1.26	-1.71	-0.57
90	-2.69	-3.558	-0.73

Table 4: For 50%, 70%, and 90% sparsity, we show the performance relative to the Frobenius norm for different magnitude-based tuple pruning criterion.

In Table 4, we show the resulting drop in top-1 accuracy relative to the Frobenius norm at three different sparsities for three alternative pruning methods. In addition to always achieving the best performance, the Frobenius norm has the additional advantage that it is defined for all Algebra variants that was consider, rather than an  $M_n(\mathbb{R})$  specific variant, for example.

## C.2 PRUNING COMPONENTS OF $M_2(\mathbb{R})$ AND $\mathbb{H}$

For  $M_2(\mathbb{R})$  and  $\mathbb{H}$ , we also prune individual tuple elements based on element norms. This equally reduces the number of non-zero weights in the network, though it does not result in entire matrix multiplies that can be skipped.

Sparsity	$M_2(\mathbb{R})$	$\mathbb{H}$
50	+0.20%	+0.18%
60	+0.37%	+0.32%
70	+0.58%	+0.49%
80	+0.93%	+0.74 %
90	+2.13%	+1.64 %

Table 5: Performance different from pruning components and entire tuples for  $M_2(\mathbb{R})$  and  $\mathbb{H}$ -AlgebraNets. Depending on the size of the network, the difference between the methods varies slightly. The main point is that pruning elements rather than tuples increases performance, more-so for higher sparsities.

In Table 5 we show the resulting increase in top-1 accuracy that results from pruning individual tuple components, rather than entire tuples. However, due to the structure  $M_n(\mathbb{R})$  and  $\mathbb{H}$  multiplication, setting individual values to 0 does not result in 0 in the output. Therefore, pruning entire tuples provides more useful computational advantages.

## D ALGEBRANET TESTS ON CIFAR

We use a network structure based on that described in Gaudet and Maida (2018). We begin with the same ResNet structure, with 128, 256, and then 512 channels in each real block. For the  $\mathbb{C}$  networks, all channel counts are divided by two. For the  $M_2(\mathbb{R})$  and  $\mathbb{H}$  networks, we assign the initial convolution, before the residual blocks, to have half the original number of channels, all other channel counts are divided by four. Thus, for  $\mathbb{H}$  and  $M_2(\mathbb{R})$  we have slightly more than 1/4 the parameters. We train with  $24 \times 24$  random crops and evaluate on  $32 \times 32$  images.

Algebra	Parameters ( $\times 10^6$ )	FLOPs ( $\times 10^6$ )	top-1
$\mathbb{R}$	3.64	32.8	94.2
$\mathbb{C}$ BN	1.85	33.6	94.2
$\mathbb{C}$ W	1.86	50.7	94.3
$M_2(\mathbb{R})$	0.94	20.7	94.3
$\mathbb{H}$ BN	0.94	41.4	93.4
$\mathbb{H}$ W	0.97	72.9	94.1

Table 6: A comparison of different AlgebraNets on CIFAR-10. BN denotes Batch Normalization, W denotes the use of whitening.

We find we are able to divide the channels in the filter by two and maintain the same performance using complex valued networks. When reducing the parameter count by a factor of  $\sim$ four, we find we are able to again match baseline performance with quaternions and  $2 \times 2$  matrices. Regularization has non-trivial effect on performance, and by more finely adjusting the  $L_2$  loss for the different algebras may result in higher top-1 accuracy. We note that the relative reduction in parameters on CIFAR-10 is not something we are able to replicate on ImageNet. The results from the main text also hold here –  $M_2(\mathbb{R})$  is the only algebra that is able to maintain accuracy while having fewer FLOPs than the baseline real network. For these experiments, we used algebra specific weight initializations, though we again verified that this does not seem to have a substantial effect.

## E EXAMPLE $M_2(\mathbb{R})$ CODE

We write the update rule explicitly for readability. Note that it is possible to concatenate the relevant terms on the channel axis to reduce the number of convolutions needed.

### CONVOLUTION

'''

*Simplified example code for  $M_2(\mathbb{R})$ .*  
*x: Input with an additional algebra axis. In the case*  
*of a convolution, either (B, H, W, C, A) or*  
*(B, C, H, W, A)*  
*w: Corresponding weight matrix, with an additional*

```

    """ algebra axis.

# Rule that describes 2x2 matrix multiplication.
mat_22_rule = [[(0, 0), (1, 2)],
               [(0, 1), (1, 3)],
               [(2, 0), (3, 2)],
               [(2, 1), (3, 3)]]

# Update each of the four algebra components.
x_new = [0, 0, 0, 0]
for i in range(4):
    for j in range(2):
        # w: weight with an extra algebra dimension.
        # x: Input with shape [B, ..., A] where A is the additional algebra dimension.
        x_new[i] += Conv2D(x[..., mat_22_rule[i][j][1],
                                w[..., mat_22_rule[i][j][0],
                                ...])

# Add bias if wanted. Add (4,) to shape.

LINEAR LAYER

# Update each of the four algebra components.
x_new = [0, 0, 0, 0]
for i in range(4):
    for j in range(2):
        # w: weight with an extra algebra dimension.
        # x: Input with shape [B, L, A] where A is the algebra dimension.
        x_new[i] += dot(x[..., mat_22_rule[i][j][1],
                                w[..., mat_22_rule[i][j][0]]))

# Add bias if wanted. Add (4,) to shape.

```

## F INCREASED ACTIVATION MEMORY

Due to the activations, there will be a slight increase in memory footprint from AlgebraNets in some cases. For example, in a  $M_2(\mathbb{R})$  AlgebraNet for ResNet-50 with channels/4, there will be C/4 convolutions performed. This would, in a naive implementation, result in twice the activation memory. However, with a properly written kernel, this would not be the case. There is, however, an additional factor: to reach comparable performance a slightly larger network than C/4 is needed. In practice about a  $1.3\times$  increase in activation memory would be incurred.