Improvement on the Quaternion-based models: extension to larger datasets and Batch Normalization

Aritra Mukhopadhyay & Adhilsha A

Goal	**To improve some quaternion models, implement models on larger datasets, and implement batch normalization.**
Datasets Used	MNIST, Cifar10 and Cifar100. (more datasets as we develop the models better)
Baseline Models	Lenet-300-100, conv2, conv4, conv6 and later more complex models like mobilenet and resnet.

What Are Quaternions?

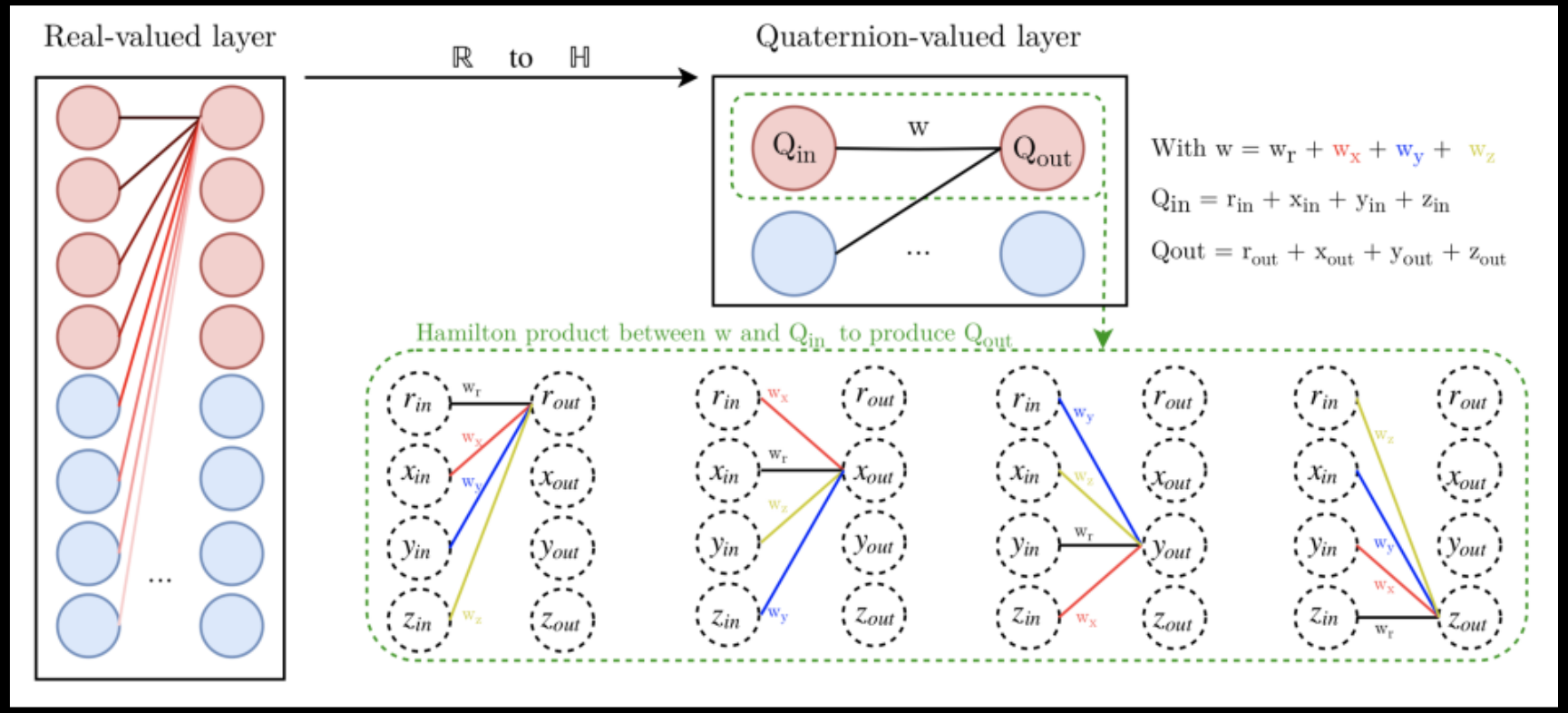
Quaternion is a four-dimensional extension of complex numbers, represented by a vector of the form $q = r + xi + yj + zk$. Given two quaternions $q1$ and $q2$, their product (known as the Hamilton product) is given by:

$$\begin{aligned} q_1 \otimes q_2 = \; & (r_1 r_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ & (r_1 x_2 + x_1 r_2 + y_1 z_2 - z_1 y_2)\,\mathbf{i} \\ & (r_1 y_2 - x_1 z_2 + y_1 r_2 - z_1 x_2)\,\mathbf{j} \\ & (r_1 z_2 + x_1 y_2 - y_1 x_2 - z_1 r_2)\,\mathbf{k} \end{aligned}$$

Quaternions multiplications not being commutative, they can be written in terms of 4*4 real matrices such that the matrix multiplication between such representations are consistent with the Hamilton product:

$$q = \begin{bmatrix} r & -x & -y & -z \\ x & r & -z & y \\ y & z & r & -x \\ z & -y & x & r \end{bmatrix}$$

Why are Quaternions used in building Neural Networks?



They can be used to make almost equally complex models with only 25% of the weights of the real version. This makes prediction whole lot more easier in low end devices.

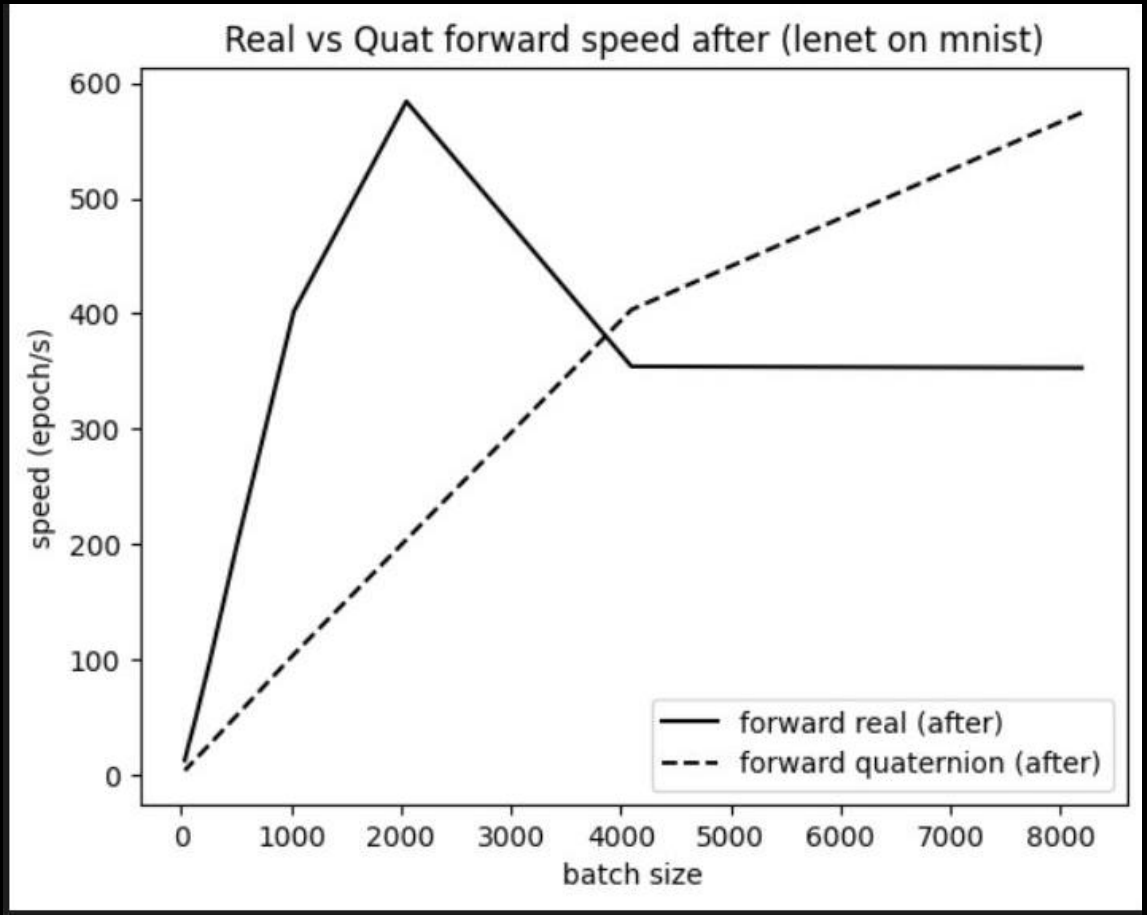
Our Approach...

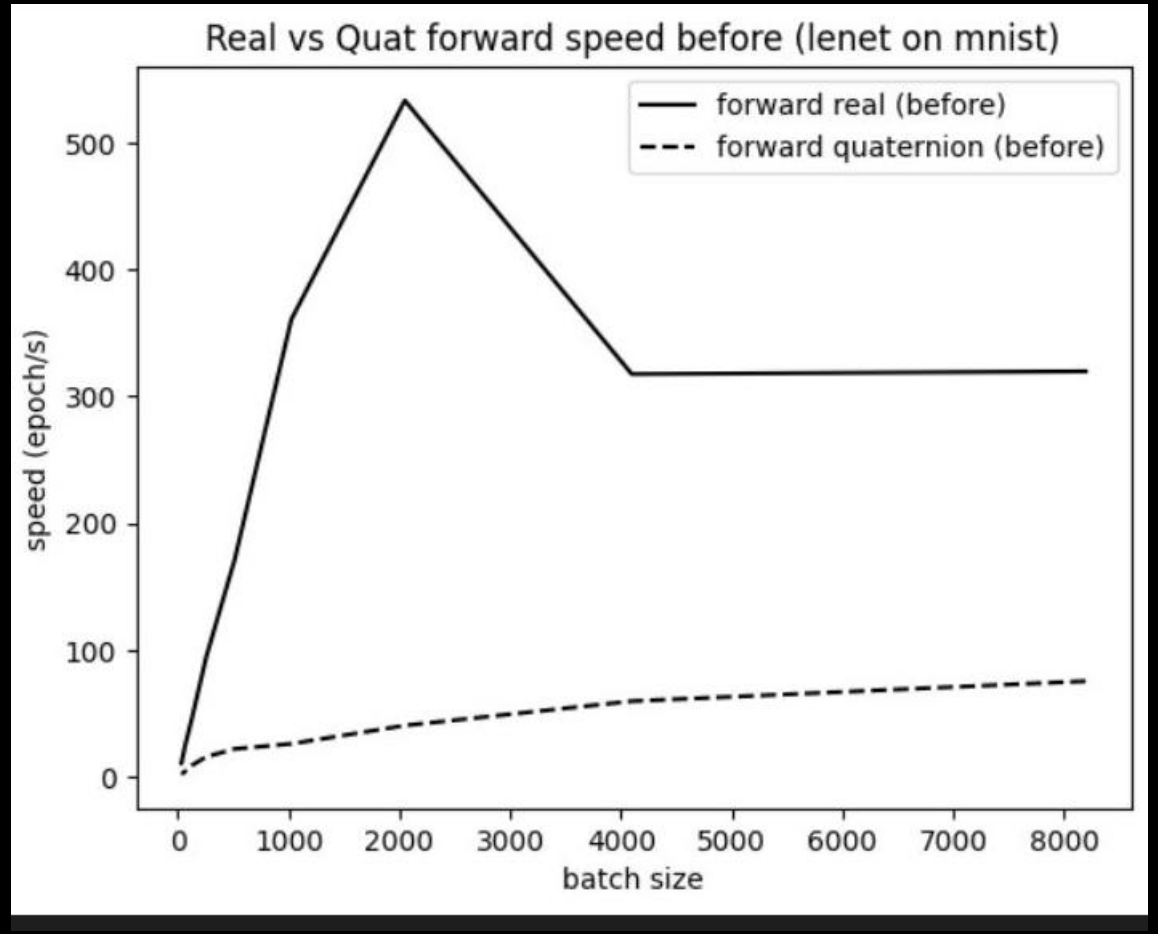
We were trying to speed up the quaternion layer calculations. For that we needed to see particularly why was it taking so long time. We found the three steps of forward propagation. They are:

1. building 4×4 Quaternion to real matrix (w)
2. Finding wx + b (applying linear function)
3. typecasting the output of step 2 to a Quaternion tensor.

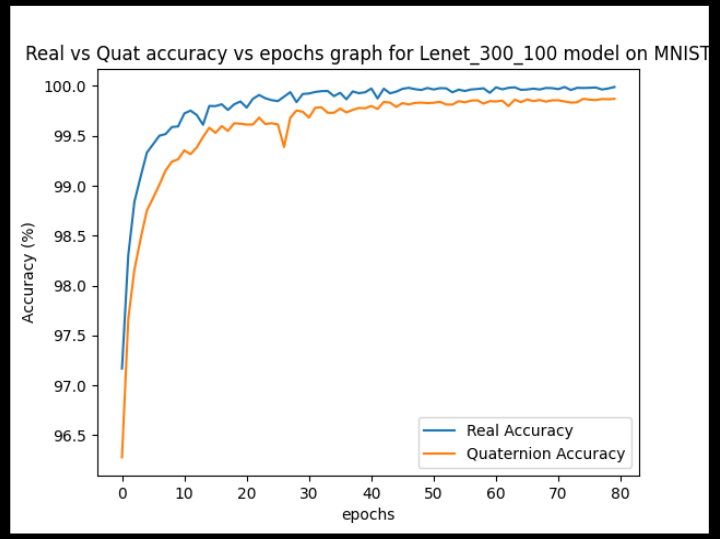
We found that step 3 was taking up 95% of the time. Further study revealed This was because of a line **q.cpu()** which was needlessly copying the x to the CPU memory (the RAM) after every layer. Being a highly experimental part of library, we changed it to **q.cuda()** and got rid of the redundant operations. This improved the speed by almost double (22.5 it/s to 57.5 it/s).

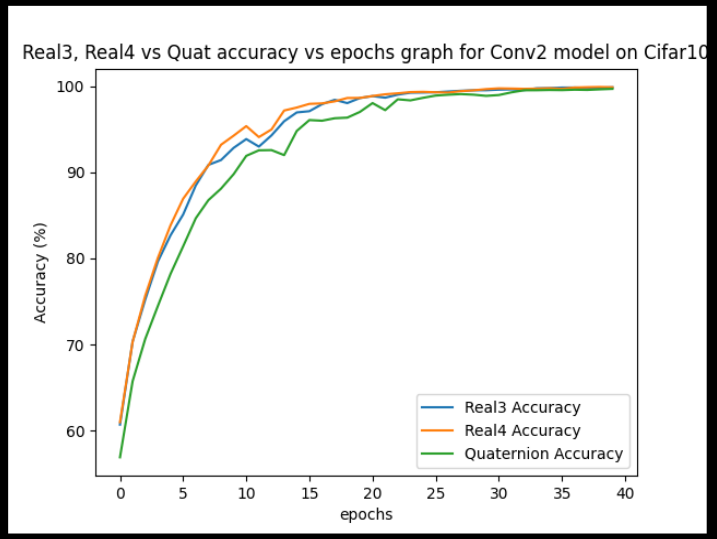
Relevant Graphs



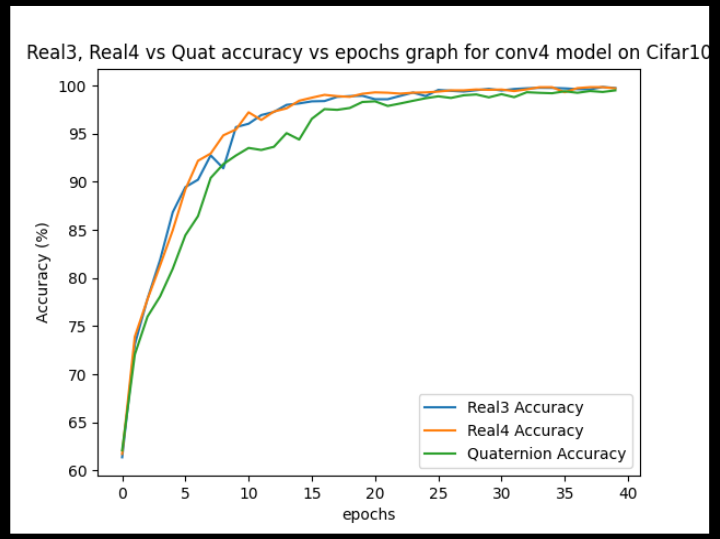


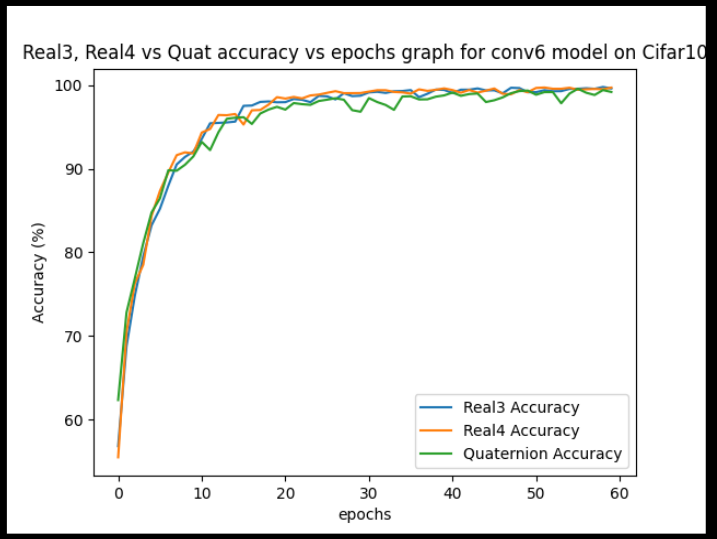
Interesting Results



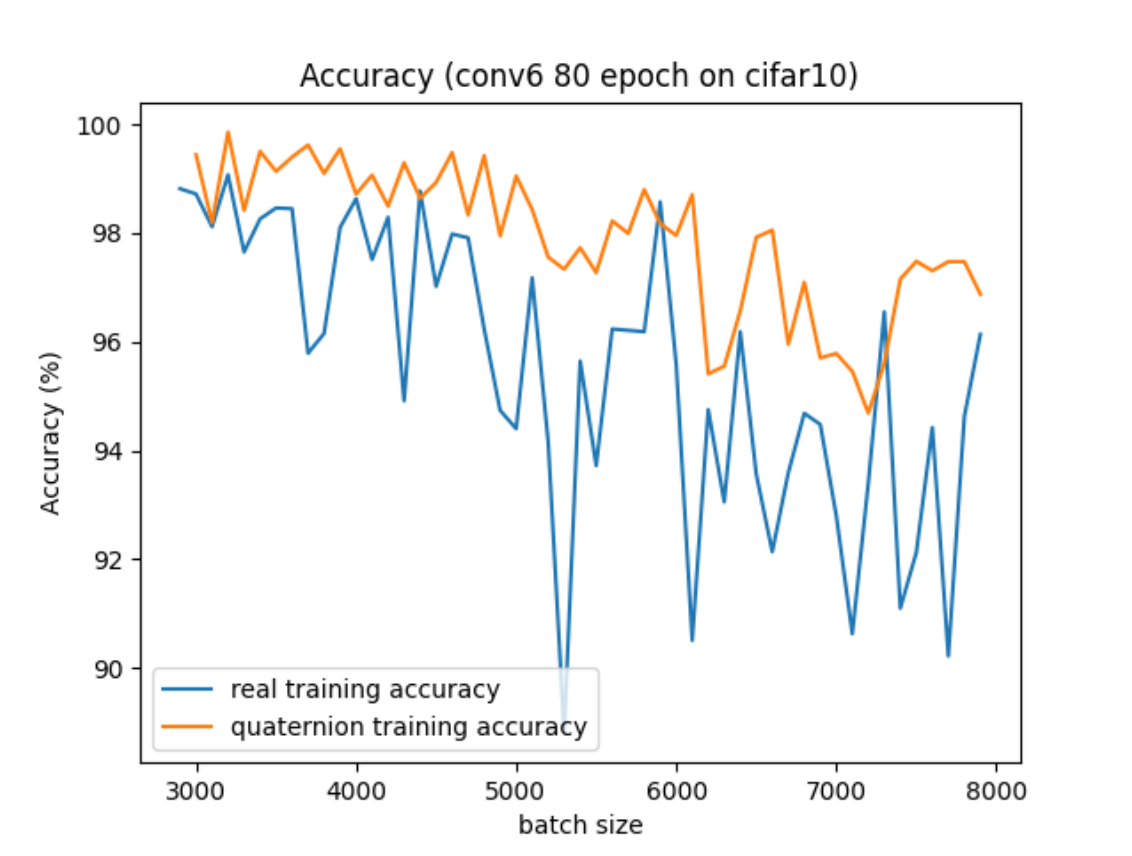


Real3, Real4 and quaternion perform similar for larger datasets.



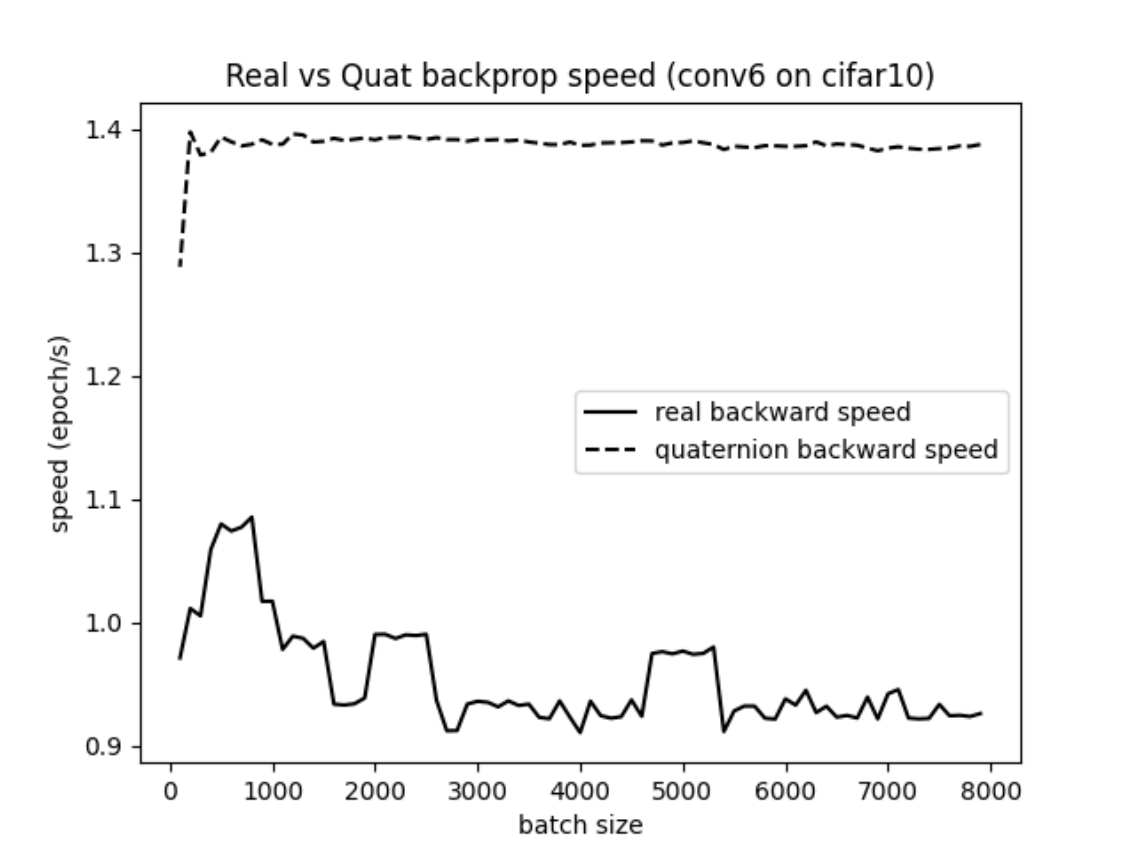


Can you explain this?



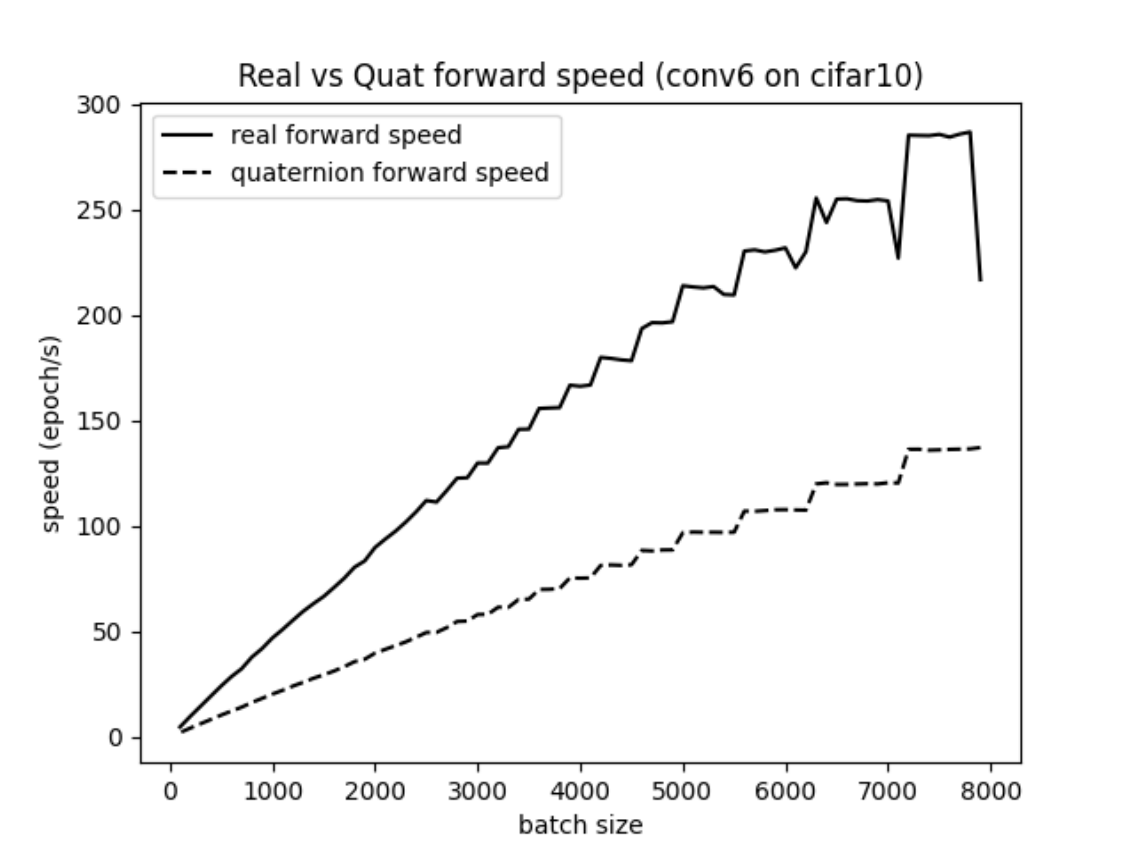
Accuracy depends on batch size

Can you explain this?



Quaternion backpropagation is almost 2x fast than real counterpart

Can you explain this?



Forward speed changes like a step function as a function of batch size