We thank all reviewers for their thoughtful comments, and respond to specific concerns below!

Reviewer 1: Note that the submission already discusses papers that carry out quantization both in the forward and backward pass ([6,15], in Lines 87-96 of our related work section). We will add the Banner et al. work to this discussion, which presents a newer 8-bit quantization scheme (at the cost of some drop in performance). We note that while our work is related, our contribution is not the quantization: indeed, we just use a simple fixed-point quantization scheme.

Instead, our contribution is in our backprop **algorithm**. As we discuss below, we believe this algorithm is both **novel** and **significant** because it limits the accumulation of error from quantization, while still delivering savings in memory.

The algorithm is novel because backprop has thus far always been implemented the same way—by using the same version of activations for forward and backward passes. Attempts at quantization have adopted this implementation and operated "locally", by only modifying per-layer operations. Breaking from this, our algorithm takes a look at the entire computation process in backprop, and shows it is beneficial to perform the forward pass exactly during training, and that it is possible to do so while still saving memory. As our analysis shows, this minimizes the effect of approximation error by preventing a cascading effect of errors building up from layer to layer.

The algorithm is significant because it allows the use of much higher rates of approximation and quantization, and thus greater memory savings. Note that in our experiments, we show that even 4-bit fixed point quantization allows successful training, even though we're using perhaps the simplest quantization function. This is precisely because we are able to do the forward-pass in full-precision. As our experiments show, doing the same quantization naively in both forward and backward passes simply fails.

In summary, the new algorithm prevents per-activation approximation errors from propagating across layers in deep networks. It allows greater levels of memory savings with even simple quantization schemes, and we believe will allow greater flexibility in exploring new kinds of approximation and quantization schemes for training.

Reviewer 2: - Effect of Activation Functions: Our algorithm is designed specifically for RELU-like activations whose gradient depends on the sign of the activation, but not the value. This is because otherwise, we would incur additional errors while computing the gradient to input layers (eq 6), which would cause errors to build up during the backward pass. Currently, it can't be applied to layers with sigmoid / tanh activations.

- Applied to RNNs: Our method can be applied to networks that have occasional sigmoid-like activations by just leaving those layers un-approximated (e.g., we don't approximate the last softmax layer in our current experiments). But RNNs and transformer networks have sigmoid/tanh activations in nearly every layer, and so the current version of our method would not work on these.

We realize that this is a bit disappointing. But as R3 also points out, ReLU-based networks cover a very large class of architectures that are widely used in many application domains. Our method will thus have real practical impact for many researchers and practitioners who train such kinds of models. Also, we believe our method can be a starting point for future work that targets RNNs, etc. Thus, we think this paper will be of interest to the NeurIPS audience.

- Densenet: If a network is fully-dense (every layer connects to every preceding layer), then our method would offer no savings. But note that DenseNets typically have sequences of dense blocks (with dense connections within blocks), and so W would be the size of the block, not the size of the entire network. Other networks use skip connections, but only from a sparse subset of layers, and would thus also have $W \ll L$ and allow for significant memory savings.

- Other optimizers: We chose momentum because this was the optimizer used by the baselines. Our method works equally well with Adam and RMSProp. As additional analysis in the revision will show (see response to R3 below), this is because the errors in gradients due to our approximation are much lower than from the randomness of SGD itself.

Reviewer 3:- More Analysis: We'll add visualizations that give a deeper explanation of why our method works: in addition to just showing training accuracy, we have computed results for the errors (when using our method vs exact training) in the actual gradients of individual layer weights. We will plot these, and compare them to the error due to SGD itself—i.e., the variance in the same gradients when computed on different mini-batches for the same model. This will show that our approximation errors are an order of magnitude lower than SGD variance, and help demonstrate why our approximation enables accurate training.

- **Other memory-saving methods:** Note that the main prior method for memory saving during training is checkpointing. This is equivalent to exact training in terms of accuracy: the disadvantage being that it's slower (the memory-speed trade-off can vary based on how frequently layers are recomputed). The other most common approach is to simply quantize / approximate activations as and when they're computed. We compare to this strategy as our 'naive quantization' baseline (for equivalent quantizations, we show this simply doesn't work).