Evaluating In-Context Learning of Inference Speed-Ups

Nelson Lojo, Derrick Han Sun, Aviral Mishra, Kesava Viswanadha

December 6, 2023

Abstract

Transformers have been shown to exhibit powerful generalization capabilities through exhibiting optimization-like behaviors in-context [1]. Improving inference time in transformers is an area of work that has seen great focus and a number of methods have looked to approximating operations in the transformer architecture to do so [3, 2, 5, 4]. We explore several possibilities for reducing computation time while attempting to minimize in-context prediction accuracy error as a result of these shortcuts. We empirically show that our optimizations can reduce inference time while still achieving comparable and even superior accuracy on select tasks. We analyze differences in performance and suggest expressivity limitations as a key cause. Our code is available at https://github.com/nelson-lojo/in-context-learning/, with model checkpoints and notebooks.

1 Introduction and Background

When inferencing with a transformer, one can use in-prompt examples that are used to make a prediction for subsequent tokens in a manner that resembles learning [7]. This phenomenon is called in-context learning. Previous work has suggested that this may be a transient effect caused by competition between encoding training distribution patterns in model parameters (in-weights learning) and encoding sequence distribution patterns in activations (in-context learning) [6]. Thus, a potential interpretation of in-context learning is as a generalization strategy, allowing models to adapt at inference time to a variety of tasks.

However, this behavior is usually observed in relatively large transformer models which often require prohibitive amounts of compute. As a result, a number of modifications to the attention mechanism have emerged to improve performance. These modifications include approximating matrix multiplications needed to construct alignment scores [4], replacing functions that require gathering [3], and changing the bit-level representation of parameters [5].

We seek to explore how the following speed-ups affect the in-context learning abilities of transformers:

1. Wortsman et al. have shown that for vision transformers, significant accuracy can be maintained when replacing the softmax activation function with ReLU after normalizing by the sequence length [3]. We expect to see a clear speed-up as ReLU does not require a gather step and is faster than computing an exponential (as in softplus, also explored in [3]).

2. Post-training Quantization in [5] has been shown to reduce both model size and speed up inference by improving memory transfer volume. The compressed representations of parameters are uncompressed (dequantized) within the GPU and operation results are sent back to the host machine as compressed Int8s. We expect any speedup to arise from saved memory transfers.

2 Methods

We replicate the training and evaluation procedures defined in [1]. Due to time constraints, we limit our analysis to only 3 of the function classes used: linear functions with input dimension 20, sparse linear functions with input dimension 20 but only 3 non-zero values, and full decision trees of depth 4 (i.e. 16 leaf nodes).

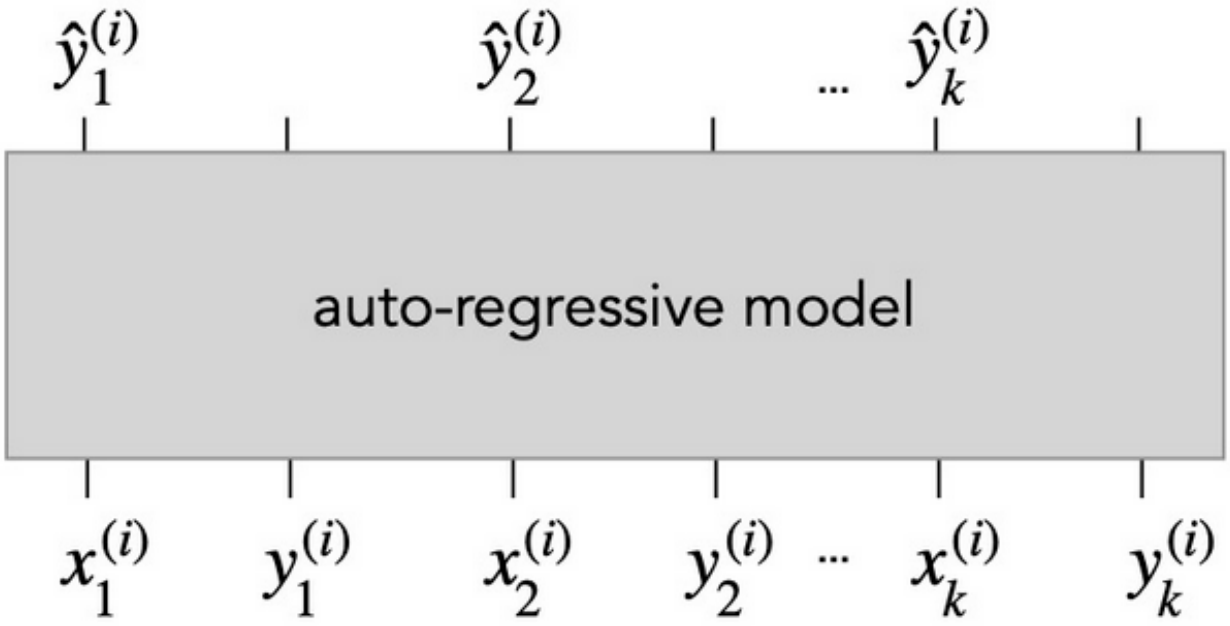


Figure 1: Portion of figure 1 from Garg et al. [1] that describes the structure of model inference.

2.1 Training

To test our optimizations, we will turn to the problem of learning a function class from in-context examples. We replicate the regression target and evaluation metric as defined in [1]. That is, for each training run, we will select a function class F from linear regression, sparse linear regression, or decision tree, then we define a $\mathcal{D}_f$ distribution over the function class and a $\mathcal{D}_x$ input domain (which is $\mathcal{N}(0, I)$ in all of our experiments). For each training sequence, we sample $f \sim \mathcal{D}_f$ and $x_1, ..., x_k \sim \mathcal{D}_x$ i.i.d.. For consistency with [1], let us also define $y_i = f(x_i)$ and let $\hat{y}_i$ be the prediction of the model at token x_i. This is illustrated in Fig. 1. As in [1], we define training loss (Eq. 1) as the sum of the squared errors between ground truth and model prediction at every valid prediction token.

$$\mathcal{L} = \sum_{i=1}^{k} (y_i - \hat{y}_i)^2 \tag{1}$$

We also accelerate training by using a curriculum, reusing the same schedule as in Garg et al. [1]. For linear regression and sparse linear regression, the number of in-context examples given to the model increases by 2 from 11 every 2000 gradient steps, stopping increasing at 41 examples until the end of training. For decision trees we start with 26 in-context examples, increasing by 5 every 2000 gradient steps, ending at 101 examples until the end of training. Training is done for 500,000 gradient steps in total in order to remain consistent with [1].

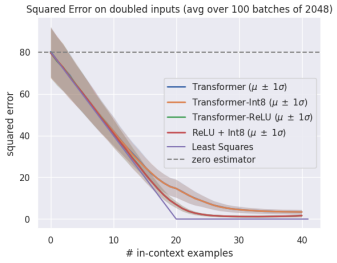
2.2 Evaluation

We replicate the accuracy evaluation metric from [1] as follows: (1) select a function class F to fetch our functions f from, (2) sample $f \sim \mathcal{D}_F$ and $x_1, x_2, ..., x_k, x_{query} \sim \mathcal{D}_x$ i.i.d., (3) feed in $x_1, f(x_1), ..., x_k, f(x_k), x_{query}$ as input sequence to the model, and (4) compute squared error between the true label y_{query} and the model's prediction $\hat{y}_{query}$.

In addition, we measure the time taken for each of the models to process batches of sequences. That is, we measure the time taken for each model to encode the token sequence, perform the forward pass through the transformer, and decode the embeddings into a prediction. We use a batch size of 2048 (limited by GPU

(a) We see that our ReLU-attention approximation has *higher* average squared error near a fully specified linear system. All models approximate least squares fairly well.

(b) ReLU-attention approximation has *lower* average squared error near a fully specified linear system. ReLU-attention imitates least squares better than the regular transformer.



(c) ReLU-attention approximation has *lower* average squared error near a fully specified linear system. Again, ReLU-attention imitates least squares better than the regular transformer.

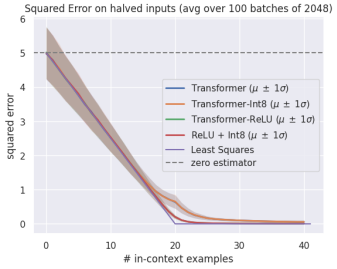


Figure 2: Linear Regression error plots for least squares, baseline, ReLU-attention, ReLU-attention quantized, and baseline quantized

Run Times (μs)		
Model Type	Run Time	# of Runs
Vanilla	5.256 ± 0.216	174
ReLU	5.148 ± 0.266	175
Int8	5.225 ± 0.217	170
ReLU+Int8	5.168 ± 0.267	176

Table 1: Average per-sequence run times of Vanilla, ReLU, Vanilla quantized (Int8), and ReLU quantized (ReLU+Int8) on a T4 GPU provisioned through Google Cloud Platform. Times are in μs. The # of runs that satisfy the cutoff described in Sec. 2.2 are included.

memory) and average this time over 200 runs, discarding runs that are greater than $Q_3 + 1.5 * (Q_3 - Q_1)$, where Q_i is the ith quartile of times.

For all evaluations, we use transformer model trained in [1] as a baseline and compare to an oracle function where appropriate (i.e. least squares for linear regression tasks).

2.3 Architectures

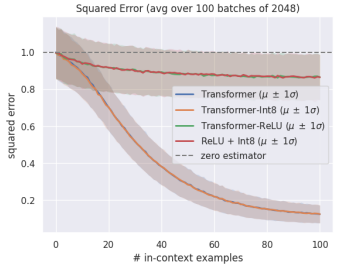
The models that we train and evaluate are identical to the model used in [1] with modifications as follows:

- **ReLU-attention** [3]: we replace the $\texttt{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$ operation in the attention layers with $L^{-1}\texttt{ReLU}\left(\frac{QK^T}{\sqrt{d}}\right)$, where L is the length of the sequence fed into the transformer. We expect this to improve inference time by eliminating the gather operation in `softmax` and thus improving parallelizability as described in [3].
- **Int8 post-quantization** [5]: we quantize all parameters into Int8 representations and dynamically dequantize parameters to perform computations in GPU registers. We expect a speedup to arise from saved memory transfers.
- **ReLU + Int8 quantization**: we combine the above two methods to qualitatively observe if quantization affects both traditional transformers and ReLU-attention transformers similarly.

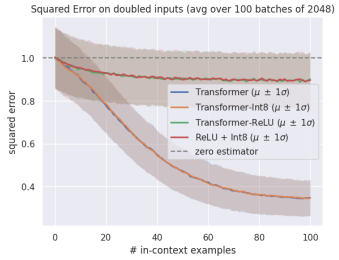
3 Results

In Table 1 we present per-sequence run times averaged over batches of size 2048. We see that no two models have a statistically significant difference, but we would like to state that the ReLU execution time averages were consistently faster than the traditional transformer executions, despite the large variances seen here.

(a) We see that although the ReLU-attention transformer is better than the zero estimator, it does worse than vanilla transformer.



(b) Again, the ReLU-attention transformer doesn't do as well as vanilla transformer. Note that both transformers were slightly negatively impacted by the change in inputs.



(c) Again, the ReLU-attention transformer doesn't do as well as the vanilla transformer. Both transformers are doing roughly as well as with regular inputs.

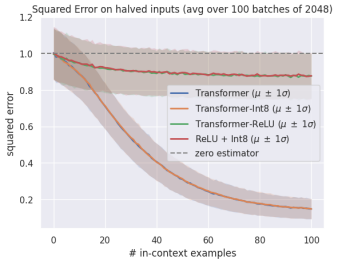


Figure 3: Decision Tree error plots for least squares, baseline, ReLU-attention, ReLU-attention quantized, and baseline quantized

We suspect that with notably more compute time, we would see the variance in executions decrease enough to verify this speedup.

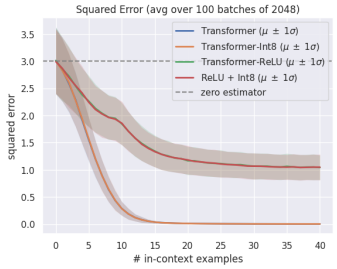
In Fig. 2a, we plot the least squares, zero estimator, and vanilla transformer, against the ReLU-attention transformer (along with quantized versions for completeness). We see that both transformer models perform with similar accuracy. In Fig. 2b, we plot the least squares, zero estimator, and doubled input vanilla transformer against the doubled input ReLU-attention transformer. Notably, in this case, we observe that a ReLU-attention transformer clearly outperforms a vanilla attention transformer from 15 in-context examples onwards. In Fig. 2c, we plot the least squares, zero estimator, and halved input vanilla transformer against the halved input ReLU-attention transformer. Like the previous case, we find that starting from around 15 in-context examples, the ReLU transformer outperforms the vanilla transformer.

In Fig. 3a, we plot the error of the zero estimator, vanilla transformer, and ReLU-attention transformer (with quantized versions as well) on the decision tree task. In Fig. 3b, we plot the error of the zero estimator, doubled vanilla transformer, and doubled ReLU-attention transformer against the number of in-context examples on the decision tree task. Finally, in Fig. 3c, we plot the error of the zero estimator, halved vanilla transformer, and halved ReLU-attention transformer against the number of in-context examples on the decision tree task. Note that all three cases, the ReLU attention transformer fares significantly worse than the vanilla transformers, but it is still better than the zero estimator.

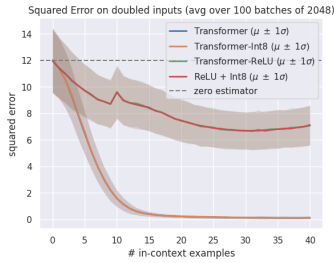
In Fig. 4a, we plot the squared error of a naive estimator and the four models with no modifications to the input. We note that the ReLU-attention transformer performs notably worse than the traditional transformer, but still exhibits some in-context learning. In Fig. 4b, we perform the same plot, but with the x_i values doubled. We notice that the irregularity around 10 examples in the ReLU-attention model's error curve sharpens and the ReLU's performance is generally much worse. By contrast, the traditional transformer is relatively unaffected. In Fig. 4c, we once again construct a similar plot, but rather than doubling x_i values, we halve them. Here, the ReLU-attention transformer does worse than it did with no scaling and the irregularity near 10 in-context examples lasts for more examples. Unlike in the linear case, the ReLU-attention transformer struggles significantly with sparse linear functions compared to vanilla attention transformers. In 4a, we see that while the vanilla transformer's error is almost 0 at around 15 in-context examples, at that same number of examples the ReLU-attention transformer's error is around 1.3. Interestingly, while we observe a convergent behavior for the former to 0 error, we see a convergent behavior for the latter to a error of 1.

Generally, we find that ReLU-attention is better than traditional transformers in robustness to scaling yet worse in accuracy for more complex funciton classes. We discuss this in more detail in Sec. 4 below.

(a) We see that the ReLU transformers fare significantly worse than regular transformers. Note the slight bump around 10 in-context examples.



(b) Again, ReLU transformers perform worse than regular transformers. This time, the bump is a spike, and we see a slight rise in error for large numbers of in-context examples.



(c) Again, ReLU transformers perform worse than regular transformers. There is a bump at 10 in-context examples, and a slight rise in error for large numbers of in-context examples.

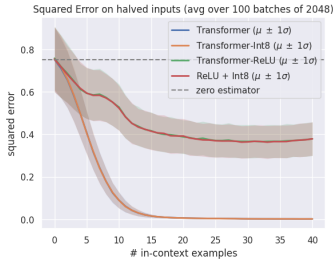


Figure 4: Sparse Linear Regression error plots for baseline, ReLU-attention, ReLU-attention quantized, and baseline quantized

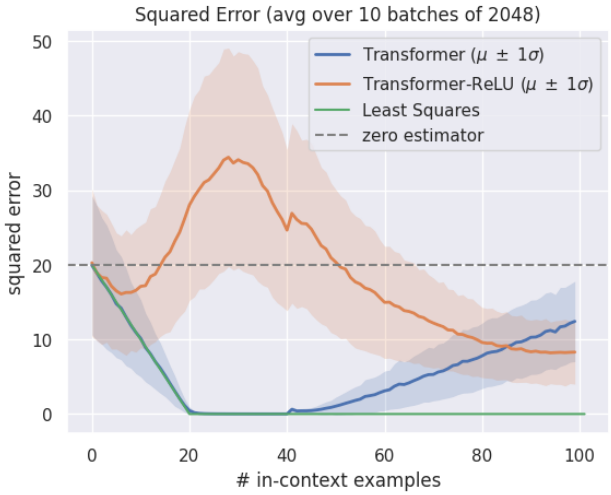


Figure 5: Error of traditional and ReLU-attention transformers on the Linear Regression task for an extended context length (100 examples) than the last curriculum phase (40 examples)

4 Discussion

4.1 ReLU-attention

We acknowledge that ReLU-attention as specified in [3] does not consider cases where the sequence length varies and that this may have affected training. We sought to affirm this understanding by extending the context window for the linear regression task and hypothesized that the ReLU-attention transformer would fail catastrophically due to drastically altered activations. We plot our results in Fig. 5 and observe that although the ReLU-attention transformer performs poorly at first, it eventually reaches *lower* error than the degrading, traditional transformer. We speculate that this may be due to the superposition effect of similar key, value, and query values for similar x_i values (and corresponding $f(x_i)$ values) in the attention calculation. We suggest a future modification to this mechanism in choosing a sequence length value that is causal – i.e. is the number of tokens the model has seen in-context at token i.

Now, we address the relation between scaling factor and squared error. Consider the attention mechanism for a vanilla transformer as written in Eq. 2. In order to better formalize our argument, let's express the scaling of the input as a hyperparameter t (temperature). Thus, we define Eq. 3 where t is the inverse scaling factor of x_i. In order to gain intuition on the effect of temperature on the value of the softmax function, we examine the edge cases. As t approaches ∞, the softmax function's output at all i approaches $\frac{1}{n}$ and the

value of x_i has less and less impact on the softmax function value at i. Thus, variance between elements of the softmax output decreases. The opposite argument can be made for t approaching 0, and we conclude that this increases variance.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}}) * V \tag{2}$$

$$Softmax(x_i, t) = \frac{e^{\frac{x_i}{t}}}{\sum_{j=1}^{n} e^{\frac{x_j}{t}}} \tag{3}$$

Let us introduce a temperature hyperparameter into ReLU as an inverse scaling factor, just as we did above with softmax: $ReLU(x, t) = max(0, \frac{x}{t})$. Examining the effect of temperature on variance of ReLU, as t approaches ∞, the output becomes more closely bound to 0, implying a decreasing variance. By the symmetrical and opposite reasoning, when t approaches 0, the output has an increasing variance.

This argument lines up with the results in Figs. 2b, 2c, 4b, and 4c, where we have more variance in $t = \frac{1}{2}$ and less variance in $t = 2$ across both the ReLU attention and vanilla transformers.

We notice that the ReLU-attention transformer consistently performs worse than the vanilla transformer and we suspect this is due to the dying neuron issue. That is, ReLU is unable to recover weights that become 0, as $\texttt{ReLU}(0) = 0$ and $\nabla\texttt{ReLU}(0) = 0$, effectively "killing" the neuron. One could argue that this takes the form of reduced expressivity for similar parameter counts.

4.2 Int8 post-quantization

We find that in all of our analyses Int8 post-quantization has no significant effect on accuracy or inference time. We propose that the lack of speedup is primarily due to the tradeoff between overhead of dequantization and memory transfer speedup at inference time, insofar that the size of our model is too small to gain value from the saved memory operations. In a similar vein, we believe our model was too small to have approximation errors compound and affect accuracy. Unfortunately due to compute constraints, we could not explore this further by examining larger models.

4.3 Linear Regression

We first seek to qualitatively understand why the ReLU-attention transformers consistently match or outperform the vanilla transformers in the Linear Regression task. Wortsman et al. briefly provide reasoning for why normalization for ReLU using the sequence length is necessary for accuracy (since this pattern-matches to the normalizing denominator for softmax and may remove the necessity of modifying other hyperparameters), and this could be extended to intuit the higher accuracy of the ReLU-attention transformer for scaled inputs [3]. That is, because ReLU preserves scaling of positive inputs, the operations surrounding the ReLU-attention mechanism may have been trained to be handle scaling as a result of "seeing" scaling frequently during training. We propose that the weight matrices preceding the ReLU-attention (replaced softmax) thus find features in the input sequence that are more robust to scaling of the inputs.

To further juxtapose and characterize the behaviors of vanilla and ReLU-attention transformers, in Fig. 6 we further examined the effects of scaling on error by plotting squared error versus input scaling factor for various in-context lengths. We find that, as expected from Garg et al. [1], both transformers approach the least squares estimate when the scaling factor is close to 1 (i.e. when $t = 1$). Interestingly, we notice that when the scaling factor is high (i.e. when t is low), both transformers approach the zero-estimator, and when the scaling factor is low (i.e. when t is high), predictions are drastically off.

We propose that this asymmetry occurs due to the combination of two effects: error due to inputs being out-of-distribution and error due to inverse scaling of noise. To formalize this, we must first acknowledge that these models use a representation of their "position" (perhaps by representing how many tokens they have seen so far). This can be seen most clearly in Figs. 2b and 2c, where the error seems to switch concavity when the number of in-context examples nears the critical point of fully specifying the linear system. With this, we can attempt to justify the behavior of the models as approximating a wholly- or semi- optimal function of the previous $x_i, f(x_i)$ pairs at each token. In particular, we believe that the predictions made by the models can be understood as least squares estimates of noisy inputs (as a consequence of finite training and thus

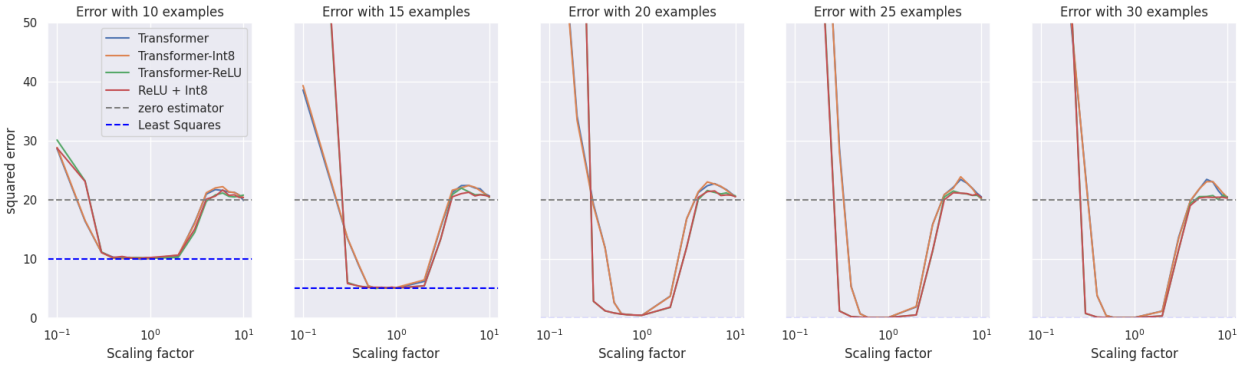


Figure 6: Squared error on the Linear Regression task as a function of scaling factor for 10, 15, 20, 25, and 30 in-context examples.

noisy bias parameters that affect the representations of input values). Thus, as we scale down the inputs, noise would eventually dominate the input signal, leading to catastrophic error. In the opposite direction, we speculate that the representations learned by the model are not well suited for values significantly outside of its training data (out-of-distribution error) and thus the model predicts similarly to the case where it has not seen any context.

4.4 Decision Trees

Now, we shall examine in-context learning on a new function class: decision trees (DTs). While the ReLU-attention transformer performs very well on learning linear functions, it struggles significantly with learning DTs, evidenced by a much shallower error curve than vanilla. In fact, it can be said that both types of transformers struggle with learning DTs, but that ReLU simply struggles more. This is shown in Fig. 3a, where we observe that even for the vanilla transformer, it takes far more in-context examples to learn a DT than a linear function. This not only agrees with Garg et al. [1], but also with intuition, where the value of a leaf node is indeterminable until you see at least one value in it – minimally requiring 30 examples at the boundaries of each bin set by the decision tree to fully determine the tree.

We repeat the scaling analysis we performed with linear regression and notice some interesting behavior. For doubled inputs, we find that the error for both types of transformers gets noticeably worse as expected, as seen in 3b. On the other hand, in 3c for halved inputs, we find that the error for both types of transformers does get worse, but the accuracy for the ReLU-attention transformer takes a significantly larger hit than the accuracy for the vanilla transformer. The error curve for the ReLU-attention transformer looks similar in both the halved and doubled case. In contrast, the error curve for the vanilla transformer in the halved case is only slightly worse than in the unscaled case, while the error curve in the doubled case is significantly worse.

In Fig. 7, we notice that the ReLU-attention transformer performs about as well as the vanilla transformer in each graph if the scaling factor for the ReLU-attention transformer is slightly greater, but for the same given scaling factor, the vanilla transformer lower-bounds the ReLU-attention transformer in terms of error. Additionally, it seems that for increasing in-context examples, the vanilla transformer error curves shift more and more to the left, meaning that they will perform better for lower scaling factors. In all cases, it seems that lower scaling factors are always extremely detrimental to accuracy – possibly due to the noisy representations presented in Sec 4.3. We also notice that for very high scaling factors, the transformers converge to almost 0 error. We will attempt to motivate this trend mathematically. Given a "high" scaling factor c, where $c > 3$:

$$x_{new} = c * x_{old}$$

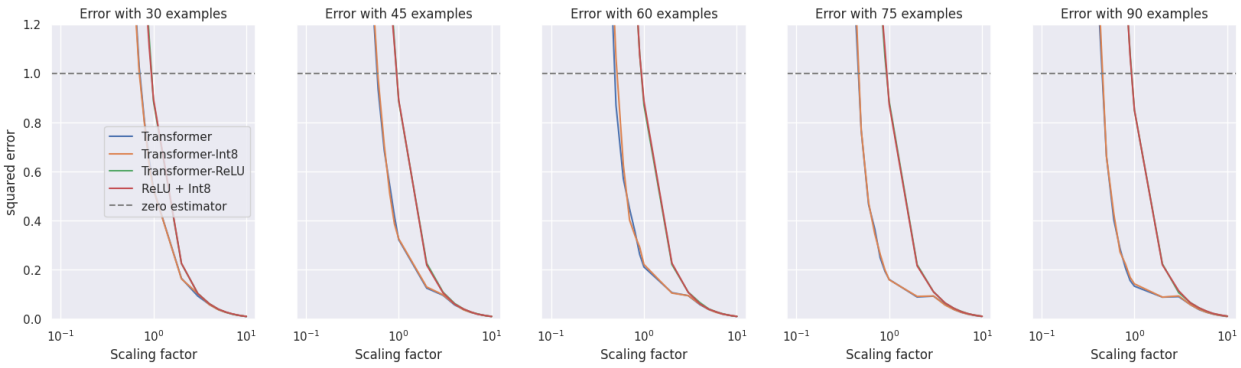


Figure 7: Squared error on the Decision Tree task as a function of scaling factor for 5, 10, 15, 20, and 25 in-context examples.

so the magnitude of is scaled up dramatically, meaning:

$$\begin{cases} x_{new} \ll 0, & \text{if } x_{old} < 0 \\ x_{new} \gg 0, & \text{if } x_{old} > 0 \\ x_{new} = 0, & \text{if } x_{old} = 0 \end{cases}$$

So, for high scaling, the input will either be very positive or very negative. Thus, functionally, the DT will "devolve" into a simple 2-category case, as prediction would become simply choosing between the categories that accept negative values that are near or below -1 or positive ones that are near or above 1. We believe that the converse should occur for $c \ll 1$, where the DT's expected value should approach the bin that encompasses input values near 0. However, we see exploding error, and we again believe this is due to noisy input representations, where scaled down inputs are more susceptible to noise in the form of biasing parameters.

4.5 Sparse Linear Regression

In Fig. 8 we see a relatively-unsurprising looking error vs scaling curve (similar to the linear case) in which scaling factors around 1 will allow, with a sufficient number of in-context examples, the vanilla transformer to reach 0 error. Although errors do not hit 0 for the ReLU-transformer case, scaling factors around 1 will still lead to the best possible error. Additionally, we notice that the ReLU model revert to the zero estimator for large scaling factors.

Another interesting observation arises from examining 4b and 4c. The vanilla transformer models' error curves are left almost unaffected by these perturbations of scale, but the ReLU-attention transformer models are clearly extremely sensitive to scale, reaching an error of about 7 at 40 in-context examples for the doubled case and about 0.35 at 40 in-context examples for the halved case. When we examine Fig. 8, we find that the conclusions we drew from the previous two graphs are substantiated. While the vanilla transformer has a "flat" error curve region centered around a scaling factor of 1 in every subplot, the ReLU-attention transformer error curve looks like a valley with a minimum slightly to the left of $scaling = 1$. We again rely on the argument of poor representations for highly scaled x_i to explain poor accuracy and the noisy input formulation to explain large error for very small scaling factors. However, this argument seems inadequate for larger counts of in-context examples, as now small scaling factors cause the ReLU-attention model to approach the zero estimator error. We suggest that this is due to conservatism arising from the low expressivity of ReLU-attention transformers. Specifically, the ReLU-attention model consistently performs poorly on the task during training, receiving a large loss despite achieving the most "correct" estimate it can produce. Consequently, we speculate that this causes the model to become conservative for regimes it cannot accurately represent. Unfortunately, we could not extend this analysis to determine if this behavior is also present in the traditional transformer due to compute constraints.

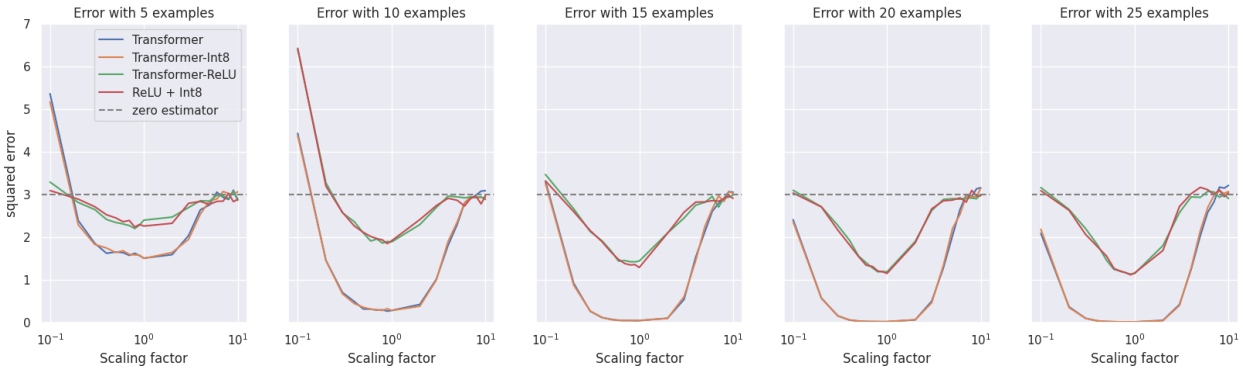


Figure 8: Squared error on the Sparse Linear Regression task as a function of scaling factor for 5, 10, 15, 20, and 25 in-context examples.

Some may find the relative inability of a ReLU-attention transformer to learn sparse linear functions to be surprising. Sigmoids are much more likely to produce outputs that are non-zero, while $\texttt{ReLU}(x)$ naturally encourages a sparse representation by setting activation output to 0 if $x < 0$. Now, we will attempt to rationalize the ReLU-attention transformer's inability. Since both models are trying to learn an underlying sparse function:

$$f(x) = s^T x$$

where s is a sparse weight vector and hence has low dimensionality. Since softmax discourages sparse representations, very few of the neurons in the feed-forward neural network die. But with ReLU as an activation function, we explicitly face the risk of encountering a dying neuron problem, ergo an expressivity issue and poor training outcome.

5 Limitations and Future Work

We acknowledge that there may be more complicated classes that are important to our investigation that we did not explore. Although we test a variety of tasks (namely linear regression, sparse linear regression, and decision trees), we could have explored more complex tasks. However, the tasks we did test showed significant performance differences between tasks of different complexity that could be further explored in future work into more complex classes.

This work was primarily limited by computational tractability. For reference, training and evaluating these three models took 200 GPU hours on modern hardware. We believe it is more reasonable to first test optimizations on tasks that are more computationally feasible that a complex task that is sure to take a great deal of resources. More complex functions would take a staggering amount of computation that are better suited to future work.

Another view of in-context learning optimization is that it may not be worth exploring the efficacy of optimizations for complex classes due to the similar computational complexity between our baseline and optimizations. While this is true asymptotically, as we do see minor improvements at this scale, particularly for ReLU-attention. We see this most strikingly affect our results with Int8 post-quantization, where our time differences are both incredibly small and not consistent.

We understand that we do not have a cutoff for whether an optimization is worth using. However, cutoff varies depending on the context. The goal of this paper is to explore the effectiveness of these optimizations, but whether their effects on inference time is worth their costs in accuracy will ultimately depend on the use-case.

The work done in this paper ties into the feasibility of transformers on less computationally capable hardware (e.g. the SCuM microchip), especially with the ReLU and quantization optimizations. Thus, additional research can be done with "small" transformers on less powerful chips.

6 Conclusions

We established the efficacy and robustness of the in-context learning abilities of ReLU-attention transformers in regressing to full-rank linear functions. Additionally, we examined the limitations of ReLU-attention when applied naively by observing the relatively subpar performance with learning sparse linear functions and decision trees. We note qualitatively – but are unable to argue – that ReLU-attention reduces inference time on average by approximately 2% in large batches and that Int8 post-quantization largely has no effect on regression prediction accuracy/inference time.

7 Acknowledgements

- Anant Sahai
- Kristofer Pister
- Yichen Liu
- Tyler Weigand
- Xiangzong Song

References

[1] Garg et al. "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes" (https://arxiv.org/abs/2208.01066)

[2] Ren at al. "Combiner: Full Attention Transformer with Sparse Computation Cost" (https://arxiv.org/pdf/2107.05768.pdf)

[3] Wortsman et al. "Replacing softmax with ReLU in Vision Transformers" (https://arxiv.org/pdf/2309.08586.pdf)

[4] Xiong et al. "Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention" (https://arxiv.org/abs/2102.03902)

[5] Zafrir et al. "Q8BERT: Quantized 8Bit BERT" (https://arxiv.org/pdf/1910.06188.pdf)

[6] Singh et al. "The Transient Nature of Emergent In-Context Learning in Transformers" (https://arxiv.org/abs/2311.08360)

[7] Brown et al. "Language Models are Few-Shot Learners" (https://arxiv.org/abs/2005.14165)