Can Transformers Learn Polynomial Regression In-Context?

Justin Kim, Tony Li, Linchuan Lu, Xiangwen Wang, Mian Wu

https://github.com/HJK-X/in-context-learning

Goal

In-context learning is the ability for a model to take in input-output pairs for a specific task and then create the output for that task given a new input [Garg et al., 2023]. In the paper "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes," this topic was explored and was found to work effectively for tasks such as linear regression. We would like to continue this study by extending in-context learning to the realm of polynomial regression to measure their efficacy in these tasks. By training a transformer on a dataset of prompts consisting of data points and a generated y value, we hope to allow the transformer to learn to predict a polynomial regression output from a new prompt.

Note: We had originally planned on doing K-Nearest Neighbors classification as well but have scrapped it as we could not find a low-runtime way to implement classification and clustering. This made training take too much time and compute, even on very low dimensional prompts.

Claim

Given a prompt of several points and their output, we claim that a transformer, trained on samples of such prompts, will be able to in-context learn polynomial function regression for low degree polynomials such as degree 2 or degree 3 polynomials but not be able to learn higher degree polynomials such as a degree 10 polynomial.

Related Works

Our paper is based on the work by Garg et al. Their study looked into how transformers could learn linear regression, sparse linear functions, decision trees, and simple neural networks without retraining the entire network. They found that prompting the transformer with several examples allowed it to learn a randomly generated function of one of these tasks. This inspired us to experiment with other types of functions that could be learned in-context.

We decided to expand the function classes from the original paper to include polynomials. This was informed by Cheng et al. in their paper, "Polynomial Regression as an Alternative to Neural Nets." Their paper explored the use of polynomial regression to explain the behavior of simple neural networks. This led to the idea of teaching a transformer how to learn polynomial regression, as Garg et al. had already shown the transformer architecture's ability to in-context learn neural networks.

Model Training

Our transformer architecture is largely adapted from the code of the source paper. We use the standard GPT-2 transformer architecture with dropout set to 0. We train the model on

11-dimensional inputs with a total of 49 in-context examples of the class we want to learn given before prediction at each training step for 40001 training steps. The model is not trained on the maximum size of inputs the whole time and is instead given gradually more complex problems with a training curriculum. Specifically, the curriculum starts with 5-dimensional inputs and 11 in-context examples fed at each step. The dimension increases by 1 and number of examples increases by 2 every 2000 training steps until they reach their maximum number as defined previously. We use this curriculum since a curriculum built like this leads to a better performing transformer [Garg et al. 2023]. Each input was generated by sampling a vector from a normal distribution with mean 0 and standard deviation 1. With these parameters, it takes approximately 1 hour and 40 minutes to train our full model using a T4 GPU on Google Colab. We could have increased our parameters by much more in order to get better trained models and possibly better results, however that would have taken too much computation and time. We use mean squared error between the true values and our predictions as our loss function during training and use squared error between the true values and our predictions for evaluation/plots.

Fixed Degree Polynomial Regression

We start off by trying to in-context learn polynomials where every term of the polynomial is of the same degree. For example:

$$f(\vec{x}) = \vec{a}^T \vec{x}^d$$

where d is the degree of the polynomial, x is an n-dimensional input vector, and a is a n-dimensional vector of constants which represent the coefficients. Note that the exponent is applied elementwise.

To generate data for this task, we sample random functions of this class by randomly generating coefficients for each term in the polynomial under a normal distribution with mean 0 and standard deviation 1. Then, a few points are generated by sampling another normal distribution with the same parameters in order to provide some in-context examples. Finally, our transformer is fed these examples and is asked to predict the output given a certain randomly generated input. This is repeated many times in order to train the transformer using the training curriculum stated at the beginning of the paper. This method is also used in order to generate the inputs and in-context examples for the plots below.

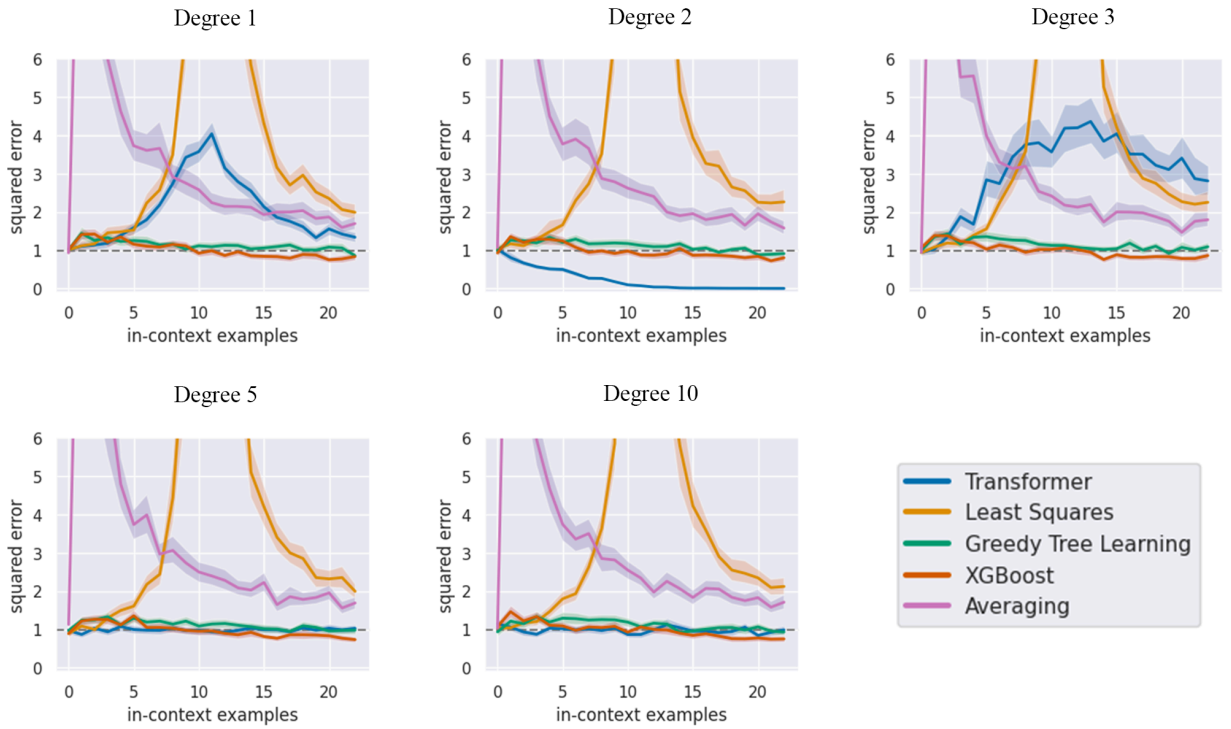

Figure 1: Graphs of the squared error of our transformer and several different metrics for specific degrees of polynomials given a certain number of in-context examples fed into the model before attempting prediction. The dashed line in the graph is the zero-estimator, which is the error you get from just guessing zero every time. All of these were generated using the same parameters for the model, with only the degree differing.

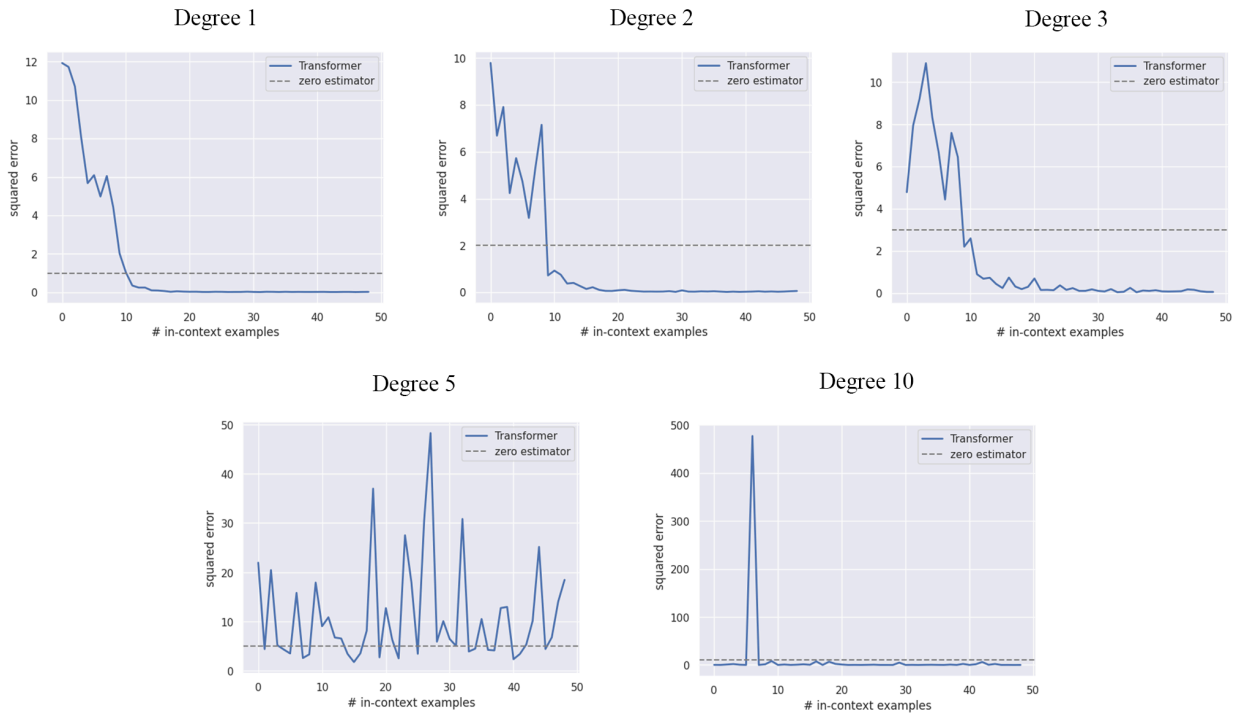

Figure 2: Graphs of the squared error of our transformer for specific degrees of polynomials given a certain number of in-context examples to predict from. The dashed line in the graph is the zero-estimator, which is the error you get from just guessing zero every time. All of these were generated using the same parameters for the model, with only the degree differing.

Note that some of the graphs in the style of Figure 1 display high variance and may not reflect actual performance. However, when the transformer is doing well, the line for the transformer will always trend towards zero, so we will focus on trends when analyzing those graphs.

In Figure 1, we can see that the transformer has a downward trend in squared error as in-context examples increase for the lower degrees (1, 2, 3) while the higher degrees (5, 10) stay stagnant and just match the performance of the zero-estimator. This tells us that for lower degrees, the transformer can learn this class of function given a few in-context examples. However, for higher degrees, feeding a lot of in-context examples fails to cause the accuracy of the transformer to increase. This implies that the higher degree systems do not extend well to the random samples which are used to make this graph, most likely because the sensitivity of the input's effect on the output of higher degree polynomials makes it harder to predict the output accurately. These trends are further reinforced by the graphs in Figure 2. In the graphs for the lower-degrees, given a sufficient amount of in-context examples (around 10-15), they will eventually go down to 0 squared-error and effectively predict the polynomial output. However, for the higher degree transformers, the value for the error is unstable. For the degree 5 graph, the error is very unstable which shows us that the transformer cannot accurately predict the output of its function class. These statements are in line with the claim which we laid out in the beginning.

Pulling From a Different Distribution

For the degree 10 graph in Figure 2, it looks like it is getting 0 squared error. However, we felt that this may be because we drew our points from a normal distribution with a standard deviation of 1. Because of this, most of our input points had a magnitude less than 1. Since the degree is so high, this would make the transformer inclined to just predict 0 for everything like the zero-estimator does since our inputs to the power of 10 would go to 0. Motivated by this idea, we decided to experiment with pulling points from a different distribution.

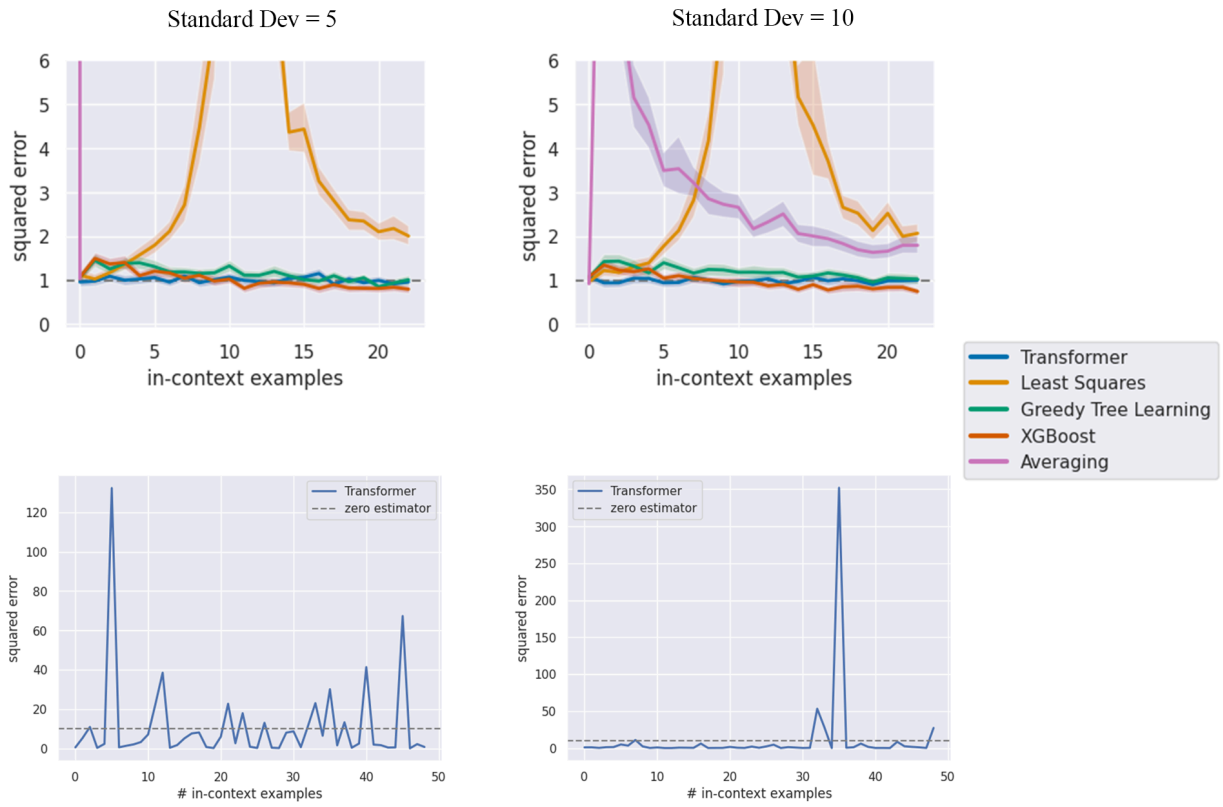

Figure 3: Squared error graphs of the same type as Figure 1 and Figure 2. These transformers are trained to predict a degree 10 polynomial. Each transformer sampled training data from a normal distribution with mean 0 and standard deviation 5 (left column) or standard deviation 10 (right column). The dashed line in the graph is the zero-estimator, which is the error you get from just guessing zero every time. All of these were generated using the same parameters for the model, with only the degree differing.

In Figure 3, two graphs in the top row are pretty similar to the graphs in Figure 1 in terms of the performance of our transformer. However, it is more interesting to look at the bottom row graphs. With a standard deviation of 5, it appears that the degree 10 polynomial provides a graph which is more similar to that of the degree 5 polynomial in Figure 2. This leads us to believe that

since a higher distribution of points leads to a higher output of the function, the transformer is led to stop guessing zeros every time. The bottom row graph with standard deviation 10 looks like the degree 10 graph in Figure 2. This result was unexpected, but we believe the transformer could be predicting a zero output while ignoring the in-context examples.

Even with a wider distribution of points, the degree 10 polynomial still does not predict very well which is in line with our claim.

Descending Degree Polynomial Regression

As our current progress looks at fixed degree polynomial functions, we would like to expand this to a polynomial with terms that have degree d down to degree 0. The general form is given by:

$$f(\vec{x}) = \vec{a}_d^{T} x^d + \vec{a}_{d-1}^{T} x^{d-1} + \ldots + \vec{a}_0^{T} x^0$$

where d is the maximum degree of the polynomial, x is an n-dimensional vector, and a_i is an n-dimensional vector with the coefficients of the degree i terms of the polynomial. Note that the exponent is applied elementwise.

We generate data in a similar manner to single-degree polynomial regression, but with a key distinction. For polynomials of varying degrees, we introduce (d+1) times the parameters compared to the single-degree scenario. This expansion accounts for each degree's individual term in the polynomial equation.

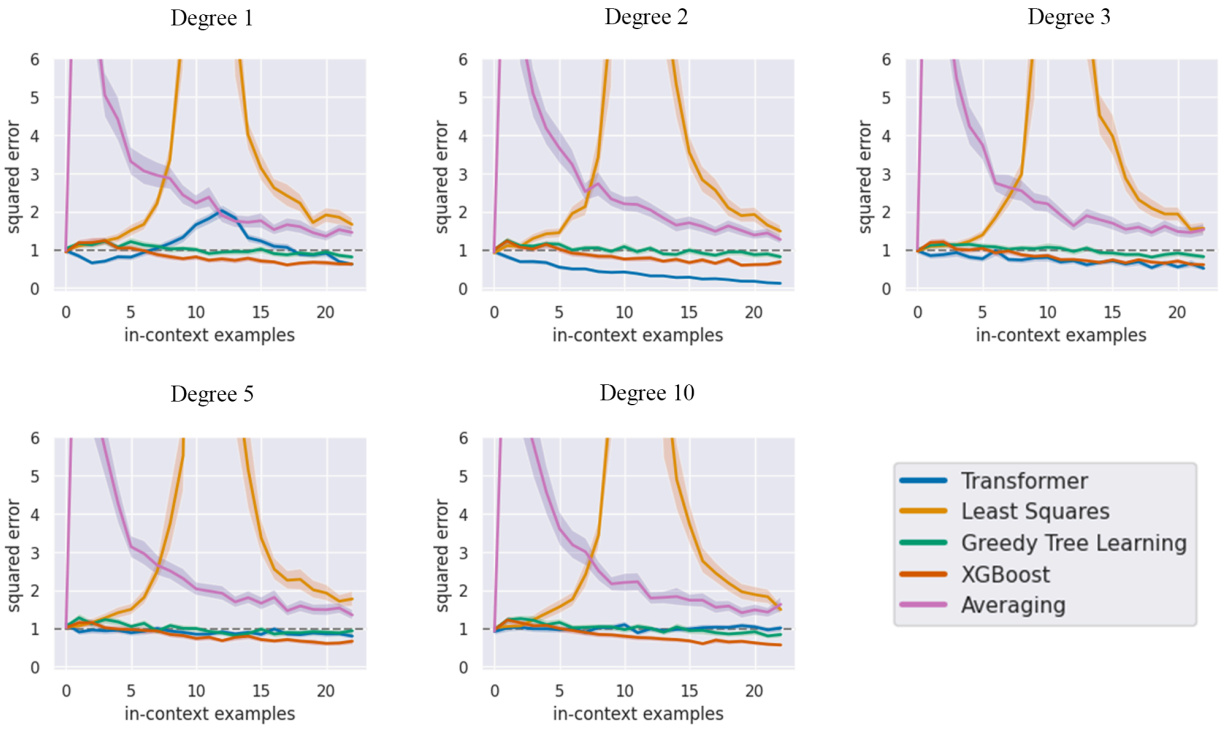

Figure 4: Graphs of the squared error of our transformer and several different metrics for specific degrees of polynomials given a certain number of in-context examples fed into the model before attempting prediction. The dashed line in the graph is the zero-estimator, which is the error if the model predicted an output of 0 every time. All of these were generated using the same parameters for the model, with only the degree differing.

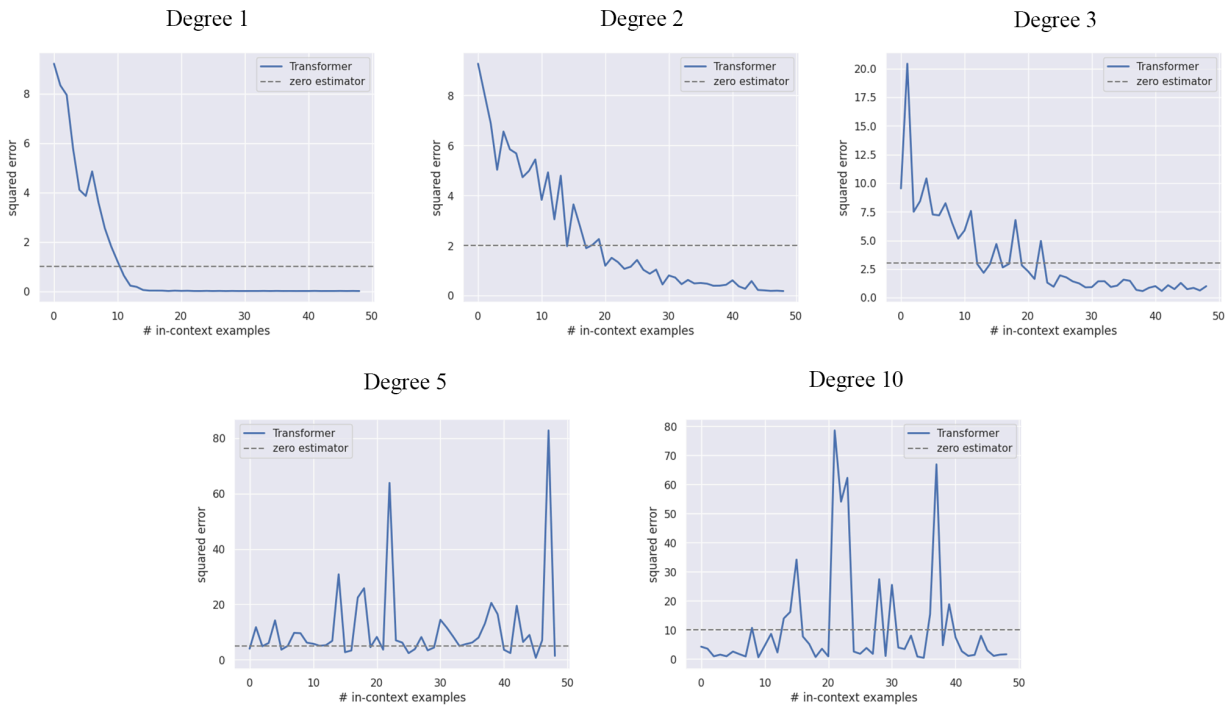

Figure 5: Graphs of the squared error of our transformer for specific degrees of polynomials given a certain number of in-context examples to predict from. The dashed line in the graph is the zero-estimator, which is the error you get from just guessing zero every time. All of these were generated using the same parameters for the model, with only the degree differing.

In Figure 3, we can again see that the transformer has a downward trend in squared error as in-context examples increase for the lower degrees (1, 2, 3). Meanwhile, the degree 5 and degree 10 graphs are still stagnant and match the zero-estimator as before. This tells us that, for lower degrees, the transformer can learn given a few in-context examples again and that for higher degrees, feeding a lot of in-context examples fails to cause the accuracy of the transformer to barely budge at all. This is reinforced by the graphs in Figure 4 in the same way as the discussion before for fixed degree polynomials as the lower-degree tasks get to 0 error while the high-degree tasks are unstable and cannot accurately predict the output of the polynomials.

It is interesting to note that it is noticeably harder for the transformer to learn the descending-degree polynomial than to learn the fixed degree polynomial. For the degree 2 graph in Figure 2, the error reached the same as that of the zero-estimator at around 10 in-context examples. Meanwhile, in the degree 2 graph in Figure 5, it takes roughly double that amount of examples in order to match the zero-estimator. This is also shown if you compare the degree 3 graphs in both figures.

Another interesting point is that for the degree 10 polynomial, we do not get a completely flat graph like the degree 10 graph in Figure 2. We suspect this is because the lower degree terms of

the polynomial contribute to the value enough so that the transformer does not attempt to be a zero-estimator (as we suspect that it does in the case of the transformer in Figure 2).

Conclusion

Transformers can in-context learn polynomial regression for lower degrees but suffer at higher degrees as indicated by our graphs.

More experiments can be done in the future in order to further test whether or not transformers can learn higher degrees than the ones we tested. For example, we were limited in compute and time so we chose our parameters such that training the model only took around an hour and 40 minutes. Someone with more compute could try to make a model with more parameters which may be able to capture the output of a higher degree polynomial. Another way to further this study would be to focus on specific subtypes of polynomials. For example, you could focus on binomial expansions of the form

$$f(\vec{x}) = (\vec{x} + \vec{a})^d$$

instead of the more general form that we used. This stricter subset may be more easily learned by the transformer.

References

Garg, S., Tsipras, D., Liang, P., & Valiant, G. (2023). *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes*.

Cheng, X., Khomtchouk, B., Matloff, N., & Mohanty, P. (2019). *Polynomial Regression As an Alternative to Neural Nets.*

Peer Review Response

We will list the questions of the peer-review which had sections which we believed needed a response. This is so we do not need to write a response to stuff such as what the goal was. Most of the changes we made to the paper was just explaining our model more and actually having experiments. We also scrapped KNN study because of what was mentioned in the beginning of the report.

What are the experiments? + the sub-questions

We changed our equation for single degree polynomials to a much simpler form as one reviewer noted we could. It is now just one vector operation instead of a summation. The reviewer who suggested this was confused about whether not all the x_i were the same or not. To clarify, in the original paper x is a random vector which is sampled from a normal distribution of mean 0 and variance 1 and x_i is element i of that vector. We have added this information into the report in response to this.

It was also unclear how many points were used to train the transformer and how we used the curriculum was vague. We have solidified our explanation of these aspects in response to this in the "Model Training" section of the paper.

We also never really explained the term zero-estimator, but now we do in the description of each figure.

One peer reviewer suggested that we take numerous samples and average them to produce better results because of how noisy our graphs were. We felt that we did not have to do this anymore since the graphs pretty clearly show trends now except for the higher degree graphs which just perform badly.

How do the experiments support the goals/claims of the paper?

Someone suggested that we add a degree-1 case as a baseline in order to compare graphs so we have done so (although it really does not serve as a good baseline for the comparing metrics graph for some reason).

Are any of the limitations discussed in the paper?

A peer reviewer suggested introducing lower degrees and focusing less on higher degrees because of how they vanish. So, we introduced the aforementioned degree 1 graph and a degree 3 graph.

What are the strengths of the paper?

Thanks

What are the weaknesses of the paper?
Everyone had obviously said that our experiments were incomplete, so we finished them.

We also now describe how the eval-time input is generated as that was mentioned in this section of a peer review.

Also, thanks for the suggestion to use the terms “fixed degree” and “descending degree” instead, as those are much better than what we had used before.

Provide a suggestion for improving the paper.
We added more methodology to our paper by clearly describing how our models were run and how data was generated.

We also simplified the function for single degree polynomials as one reviewer had suggested.

Someone suggested that we tackle smaller degrees so we added some more plots of smaller degrees.

The suggestion that we use a different function family was interesting but we felt that our graphs looked pretty good with our current family so we stuck with it. Thanks for the interesting idea though.

Are all the plots in the paper clearly interpretable with well-defined and explained axes, with methodology clearly explained in the paper text?
For the graphs, # in-context examples means how many in-context examples of the function class were given to the trained transformer before it tried to predict. This was added in the description of the figures in order to avoid confusion. Also, averaging is one of the metrics that we put on the plot in order to compare our transformer to.

Old Self-Review For Draft Report (Not Sure If We're Supposed to Leave This In)

What is the main goal of the project?

The main goal of this project is to test how well a transformer can in-context learn polynomial regression and KNN classification problems.

What are the main claims?

The claims were that a transformer will be able to in-context learn the tasks of polynomial regression and k-nearest neighbors classification.

What are the experiments?

The results of the experiments are evaluated by using the squared error and comparing the score of the transformers with the score given by other metrics such as Least Squares. The data is generated by randomly sampling a 20-dimensional vector from a normal distribution with mean 0 and variance 1. The only task so far is polynomial regression where every term has the same degree.

How do the experiments support the goals/claims of the paper?

The experiments fail to support the goals and claims of the paper. The graphs for single-degree polynomials show that at best, it performs as well as the trivial case of a zero estimator. The experiments show the process of work toward actually having good code that can properly test the goal which is put forward by this paper.

Are any of the limitations discussed in the paper?

A limitation discussed in the paper is exponential decay of the input. This was addressed in the TODO section, but the paper has not completely decided on the variance of the sampling function.

What are the strengths of the paper?

A strength of the paper is that all of the code is in notebooks. This makes it easy to prototype and run the experiments. There was a decent amount of exploration within the task of single-degree polynomial regression, with several degrees tested.

What are the weaknesses of the paper?

A weakness of this paper is that it is not very complete. While it starts to tackle the goal of the paper somewhat, not much data has actually been collected to sufficiently support any claim. Another weakness is that it does not utilize enough outside literature to support its ideas. It focuses on using the information from one paper for right now rather than a variety of papers.

Is the paper reproducible?
The experiments can be rerun with a notebook in the GitHub repo provided. The results are not exactly the same as the graphs in the report, however they portray similar results. Also, a provided notebook runs through a toy example of the paper as well as generating the graphs in the paper through the use of the weights generated when running the notebook to originally create the graphs. Although, the code to generate the graphs for the experiment using higher variance is missing.

Are all the plots in the paper clearly interpretable with well-defined and explained axes, with methodology clearly explained in the paper text?
The plots are clearly interpretable and the axes are always explained in the caption underneath the plot. There is a legend which explains the different lines in the plot.

Is the English in the paper correct and clear?
Yes, it is correct and clear.

Do you have any feedback on any TODOs that the authors have left at this stage?
You just need to finish writing the code and test it. Maybe look at some papers to determine some additional methods of generating baselines for these tasks.