What Can Transformers Learn In-Context? A Case Study of Simple Function Classes

Anonymous Author(s)
Affiliation
Address
email

Abstract

In-context learning is the ability of a model to condition on a prompt sequence consisting of in-context examples (input-output pairs corresponding to some task) along with a new query input, and generate the corresponding output. Crucially, in-context learning happens only at inference time without any parameter updates to the model. While large language models such as GPT-3 exhibit some ability to perform in-context learning, it is unclear what the relationship is between tasks on which this succeeds and what is present in the training data. To investigate this, we consider the problem of training a model to in-context learn a function class (e.g., linear functions): given data derived from some functions in the class, can we train a model (e.g., a Transformer) to in-context learn most functions from that class? We show empirically that standard Transformers can be trained from scratch to perform in-context learning of linear functions—that is, the trained model is able to learn unseen linear functions from in-context examples with performance comparable to the optimal least squares estimator. In fact, in-context learning is possible even under two forms of distribution shift: (i) between the training data of the Transformer and inference-time prompts, and (ii) between the in-context examples and the query input during inference. We also show that we can train Transformers to in-context learn more complex function classes: sparse linear functions where the model outperforms least squares and nearly matches the performance of Lasso, and two-layer neural networks where the model performs comparably to neural networks trained on in-context examples using gradient descent.

1 Introduction

Large language models, such as GPT-3 [7], are able to perform *in-context learning*: given a prompt containing examples from a task (input-output pairs) and a new query input, produce the corresponding output. For example, these models are able to produce English translations of French words after being prompted on a few such translations or add numbers based on examples of addition, e.g.:

$$\underbrace{\text{maison} \to \text{house},\ \text{chat} \to \text{cat},\ \text{chien} \to}_{\text{prompt}}\ \underbrace{\text{dog}}_{\text{completion}}\ .$$

This capability is quite intriguing as it allows models to adapt to a wide range of downstream tasks on-the-fly—i.e., without the need to perform any parameter updates after training [7, 19, 36, 6]. However, it is unclear to what extent these models have developed the ability to learn *new tasks* from in-context examples alone as opposed to simply indexing into a vast set of known tasks from the training data [48, 25].

The goal of our work is to empirically understand in-context learning in a well-defined setting. Concretely, we frame in-context learning as the ability to learn a *function class* from in-context

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

examples. That is, we say that a model can in-context learn a function class $\mathcal{F}$ if, for "most" functions $f \in \mathcal{F}$, the model can approximate $f(x_{\text{query}})$ for a new query input x_{query} by conditioning on a prompt sequence $(x_1, f(x_1), \ldots, x_k, f(x_k), x_{\text{query}})$ containing the in-context examples and the query input.

Formally, let $D_{\mathcal{X}}$ be a distribution over inputs and $D_{\mathcal{F}}$ be a distribution over functions in $\mathcal{F}$. A prompt x_{prompt} is a sequence $(x_1, f(x_1), \ldots, x_k, f(x_k), x_{\text{query}})$ where inputs (i.e., x_i and x_{query}) are drawn i.i.d. from $D_{\mathcal{X}}$ and f is drawn from $D_{\mathcal{F}}$. We say that a model M can in-context learn a function class $\mathcal{F}$ up to ϵ, with respect to $(D_{\mathcal{F}}, D_{\mathcal{X}})$, if it can predict $f(x_{\text{query}})$ with an average error:

$$\mathbb{E}_{x_{\text{prompt}}}\left[\ell\left(M\left(x_{\text{prompt}}\right), f\left(x_{\text{query}}\right)\right)\right] \leq \epsilon, \tag{1}$$

where $\ell(\cdot, \cdot)$ is some appropriate loss function, such as the squared error. Note that, here, being able to in-context learn a function class is a property of model M alone independent of how it was trained.

Within this framework, we can now concretely ask:

What classes of functions can be in-context learned by a given model?

We are interested not only if a given model M can in-context learn a function class, but how one might train a model M to have that ability. In particular, we focus on Transformer models [46, 34]—the architecture behind recent large language models—trained from scratch to perform in-context learning for simple, well-defined function classes (e.g. linear functions). Our findings are as follows.

Transformers can in-context learn linear functions. We show empirically that we can train a standard Transformer from scratch to in-context learn the class of linear functions in 20 dimensions (see Figure 1 for an illustration). That is, the trained model is able to in-context learn unseen linear functions at inference time with performance comparable to the optimal least squares estimator. Thus, the Transformer itself encodes an effective learning algorithm which is competitive with least squares.

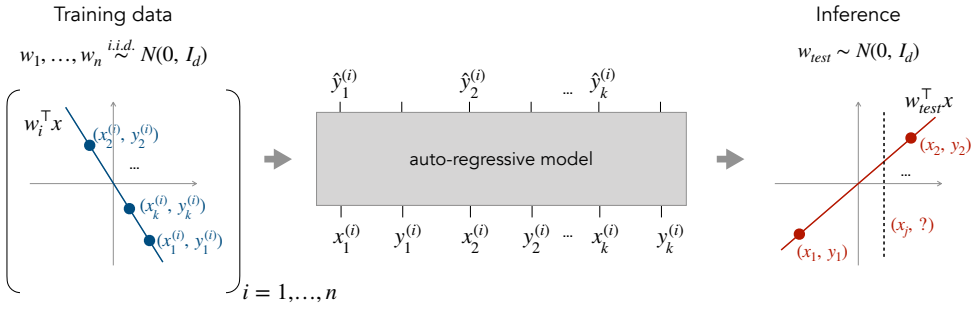


Figure 1: *Can we train a model that in-context learns a function class (here linear functions)?* We train a Transformer by repeatedly sampling a random function f from that class, as well as random inputs $x_1, \ldots, x_k$ and training the model to predict each $f(x_i)$ given the prompt $x_1, f(x_1), \ldots, x_{i-1}, f(x_{i-1}), x_i$ (using the squared loss). Then, during inference, we evaluate the model's ability to predict accurately on new, *unseen* functions.

Generalization to out-of-distribution prompts. To understand to what extent the trained Transformer has learned a "real algorithm" for linear regression, we consider in-context learning under two types of distribution shifts: (a) a shift between the prompts encountered during training and inference (e.g., training on prompt inputs from an isotropic Gaussian but testing on inputs from a skewed Gaussian), (b) a shift between the in-context examples and the query input during inference (e.g., in-context examples lie in one orthant and the query input lies in another). We find that the in-context learning performance of the model is quite robust to such shifts, suggesting the model has indeed learned to approximately perform linear regression to to a surprising degree.

Beyond linear regression. We also consider the function classes of 3-sparse linear functions and two-layer ReLU neural networks with 100 hidden units and show that we can train models that can in-context learn these classes. For sparse linear functions, the trained model is able to exploit sparsity, obtaining performance better than least squares and comparable to Lasso. For neural networks, the corresponding model is able to obtain performance comparable to neural networks trained using gradient descent on in-context examples, and is also able to in-context learn linear functions. Note that learning these function classes requires involved algorithms (e.g., gradient descent with the Lasso objective), and our result suggests that a Transformer can discover a valid algorithm during training.

Role of model capacity and problem dimension. Finally, we explore how the ability of a model to in-context learn linear functions scales with model capacity and the dimensionality of the problem. We find that increasing the capacity of the model improves performance significantly. Moreover, increasing the capacity often significantly improves out-of-distribution generalization, even when the absolute improvement in the standard error is small.

2 Training models for in-context learning

Here, we describe a general methodology for training a model to in-context learn a function class $\mathcal{F}$. To do so, we start by constructing training prompts as follows. For each prompt, we first sample a random function f from the class according to a distribution $D_{\mathcal{F}}$, then sample a set of randomly chosen inputs $x_1, \ldots, x_{k+1}$ independently from a distribution $D_{\mathcal{X}}$, and finally evaluate f on these inputs to produce the prompt $x_{\text{prompt}} = (x_1, f(x_1), \ldots, x_{k+1}, f(x_{k+1}))$. For example, in the case of linear functions, inputs could be drawn from the isotropic Gaussian distribution $N(0, I_d)$, and a random function chosen by sampling weights w from $N(0, I_d)$ and setting $f(x) = w^\top x$.

Now, given this set of prompts, we want to train a model to predict $f(x_i)$ for a given x_i based on a set of in-context examples. Concretely, let x^i_{prompt} denote the prompt prefix containing i in-context examples (the first i input-output pairs) and the $(i+1)^{\text{th}}$ input: $x^i_{\text{prompt}} = (x_1, f(x_1), x_2, f(x_2), \ldots, x_i, f(x_i), x_{i+1})$. Then, given a model M_θ parametrized by θ, we train that model by minimizing the expected loss over all the prompt prefixes up to size k:

$$\min_\theta \; \mathbb{E}_{x_{\text{prompt}}} \left[\frac{1}{k+1} \sum_{i=0}^{k} \ell \left(M_\theta \left(x^i_{\text{prompt}} \right), f\left(x_{i+1} \right) \right) \right], \tag{2}$$

where $\ell(\cdot, \cdot)$ is an appropriately chosen loss function. We do so through stochastic gradient descent where, at each training step, we sample a random prompt (corresponding to a random function from $\mathcal{F}$), and take a gradient step to improve the loss averaged over all prefixes of the prompt.

Curriculum learning. Many natural function classes contain functions of varying complexity. We exploit this by training our model via curriculum learning [3, 13, 41, 47], where we start by training on an easier subset of functions (e.g., linear functions in a 20 dimensional space with weight vectors lying in a fixed 5 dimensional subspace), and slowly increasing the function complexity over training. We find that this speeds up training drastically, often allowing us to train models that we could not without curriculum learning (see Section 6 for more details).

3 In-context learning of linear functions

In the previous section, we discussed a general methodology for training models to in-context learn a class of functions. We will now instantiate this framework by focusing on Transformer models [46] trained to in-context learn the class of linear functions.

Prompt distribution. We consider the class of linear functions on d-dimensional vectors $\mathcal{F} = \left\{ f \mid f(x) = w^\top x,\ w \in \mathbb{R}^d \right\}$, where $d = 20$. We sample $x_1, \ldots, x_k$, x_{query}, and w independently from the isotropic normal distribution $N(0, I_d)$. We then compute each $y_i = w^\top x_i$ and construct the prompt as $x_{\text{prompt}} = (x_1,\ y_1,\ x_2,\ y_2, \ldots,\ x_k,\ y_k,\ x_{\text{query}})$.

Model training. We use an decoder-only Transformer architecture from the GPT-2 family [35] with 12 layers, 8 attention heads, and 256 dimensional latent embedding space. We map each prompt point (corresponding to a prompt input or output) into the latent embedding space of the Transformer through a (learnable) linear transformation and use a (learnable) linear head to produce a scalar output. We train the model on prompts containing $2d$ in-context examples through the training objective in (2) using the squared error as a loss function ℓ. We use a batch size of 64 and train for 500k total steps. Note that this architecture allows us to compute the prediction $(M_\theta(x^i_{\text{prompt}}))$ for all prefixes of the prompt in a single forward pass. We provide further training and architecture details in Appendix A.

Baselines. To contextualize the performance of our Transformer model, we compare it to a few other approaches: (a) the least squares estimator which computes the minimum-norm linear fit to the in-context examples $(x_i,\ y_i)$, (b) k-Nearest Neighbors where we average the y_i values for the k nearest neighbors to x_{query}, (c) averaging the values $y_i x_i$ to obtain an estimate for w and compute

the inner product of this estimate with x_{query}. Least squares is the optimal estimator for this problem and thus serves as a lower bound to the best error one can achieve. The other two baselines are consistent (but sub-optimal) estimators that are significantly easier to compute and thus provide us with estimates of what error simple approaches would obtain. See Appendix A.3 for more details.

We show the in-context learning ability of the resulting model along with the relevant baselines in Figure 2. The model is able to in-context learn random linear functions, performing comparably to the optimal least squares estimator for all prompt sizes. While the simpler baselines achieve non-trivial error, they are far worse, indicating that the trained model is encoding a more complex algorithm.

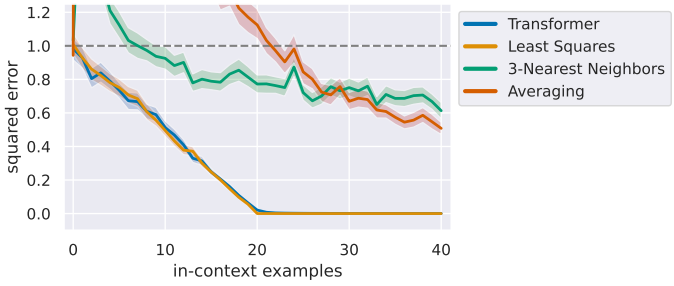


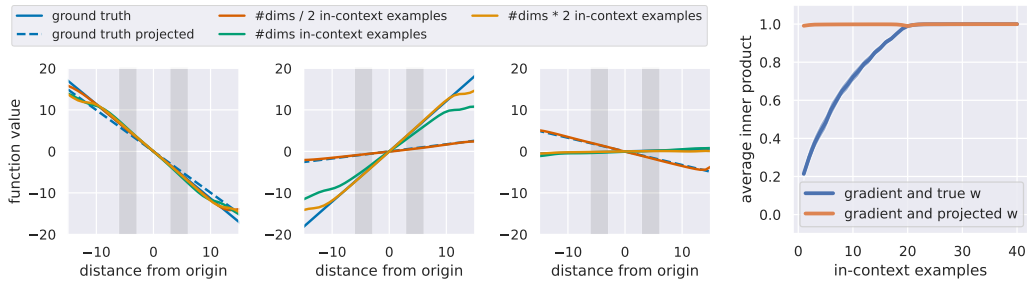
Figure 2: *Evaluating the ability of the trained Transformer to in-context learn linear functions.* We plot the normalized squared error of the model $((M(x_{\text{prompt}}) - w^\top x_{\text{query}})^2/d)$, along with the relevant baselines, as a function of the number of in-context examples. The error of the Transformer decreases at a rate comparable to least squares. When the number of in-context examples reaches the problem dimension d (20 in this case), least squares achieves 0 error while our model achieves an error of 0.02 which improves to 0.0006 at $2d$ in-context examples. While the simple baselines obtain better-than-trivial error (dashed line), their performance is quite poor. (The error is averaged over 1280 prompts with 95% confidence intervals over 1000 bootstrap trials.)

Is the model generalizing or just memorizing? One might wonder if the model is actually able to in-context learn unseen functions or whether it is predicting by memorizing the prompts seen during training. First, note that the probability of the model encountering a training prompt similar to the one used for testing is astronomically low—the prompt inputs alone lie in a 400-dimensional space when predicting with $2d$ in-context examples ($d = 20$). Moreover, even considering the possibility that the model encountered a similar *weight vector* during training cannot explain the model's performance. That is, the model encounters 32 million random weight vectors during training and even using best of these vectors would lead to an expected error of around 0.2 (computed empirically). However, the model is able to achieve an error less than 0.001 for a prompt with $2d$ in-context examples. Further, in Section 6, we show that the model is able to obtain a similar error even when trained on prompts generated using only $10,000$ distinct weight vectors, in which case using the best weight vector seen during training would yield an even worse error of around 0.5. Thus we can conclude that the model has developed the ability to perform in-context learning of unseen linear functions, and cannot be relying on memorization of the training prompts or the weight vectors used to generate them.

3.1 Probing the function that we learn in-context

Recall that the goal of our model is to output $w^\top x_{\text{query}}$ when given the prompt $x_{\text{prompt}} = (x_1,\ w^\top x_1, \ldots, x_k,\ w^\top x_k,\ x_{\text{query}})$. Thus, if we fix the prefix given by the k in-context examples, we can view the output of the model $M(x_{\text{prompt}})$ as a function $\hat{f}_{w,x_{1:k}}(x_{\text{query}})$, that aims to approximate the ground truth $w^\top x_{\text{query}}$. When $k < d$, the ground truth cannot be recovered accurately and our model should approximate $(\text{proj}_{x_{1:k}} w)^\top x_{\text{query}}$, where $\text{proj}_{x_{1:k}} w$ is the projection of w onto the subspace spanned by $x_1, \ldots, x_k$. Here, we investigate how well our model achieves this goal.

Visualizing along a random direction. For a randomly sampled prefix, we visualize how $\hat{f}_{w,x_{1:k}}(x_{\text{query}})$ behaves as we vary the query input along a random direction. That is, we pick a random unit vector x, and evaluate $\hat{f}_{w,x_{1:k}}(\lambda x)$ as we vary λ, the distance of the query input from origin—cf. Figure 3a. We observe that $\hat{f}_{w,x_{1:d}}(\lambda x)$ and $\hat{f}_{w,x_{1:2d}}(\lambda x)$ closely match the ground truth and $\hat{f}_{w,x_{1:d/2}}(\lambda x)$ matches the projected ground truth, when the distance from origin is not too large



(a) visualizations (b) gradients

Figure 3: *Probing the prefix-conditioned function*. (a) We plot the model prediction as we fix the in-context examples and vary the query input along a random direction (for three random prompts). The shaded region denotes the interval in which the norm of a randomly sampled input lies with probability 0.99. When the scale of the query input is not too off, the model prediction matches the ground truth linear function, given d or 2d in-context examples. Given d/2 in-context examples, it matches the linear function obtained by projecting ground truth w on the subspace spanned by example inputs. (b) We compute the gradient of the model prediction with respect to the query input, and plot its (normalized) inner product with the true w and projected w, averaged over multiple query inputs. The gradient aligns almost perfectly with w for number of in-context examples $k \geq d$, and with projected w for all k, indicating that the model prediction aligns locally with the ground truth.

compared to the norm of a typical randomly sampled input. In Appendix B.1, we show that the model is quite robust to scaling the query input: the error doesn't increase much as we scale up the query input by a factor of up to 2, or scale down by a factor of up to 16, and degrades slowly after that.

Local correctness. So far, we have seen that the model is able to make predictions close to the ground truth for randomly drawn query inputs. We will now turn our attention to the local change of $\hat{f}$ around x_{query} by considering the gradient of the function $\hat{f}_{w,x_{1:k}}(x_{\text{query}})$ with respect to x_{query} (our model is fully differentiable so we can compute the gradient directly). In Figure 3b, we show the inner product between the gradient and the projected ground truth weight $(\text{proj}_{x_{1:k}} w)$ (both normalized), averaged over multiple random query inputs, and observe that they align almost perfectly. When $k \geq d$ the gradient perfectly aligns with the ground truth w itself.

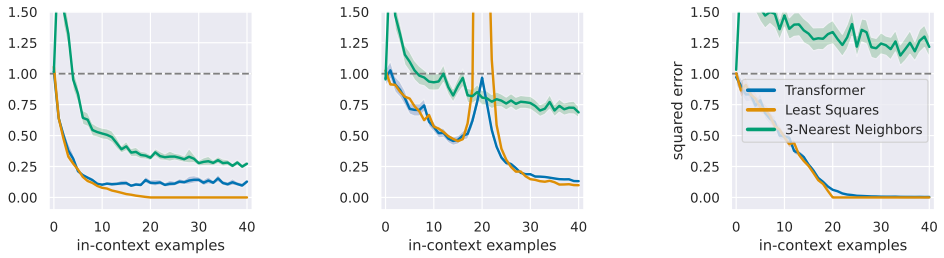
4 Extrapolating beyond the training distribution

In the previous section, we trained a model that can in-context learn linear functions with respect to the distribution of prompts encountered during training. In this section, we examine whether the model has learnt to perform linear regression from in-context examples more generally, by evaluating its performance on out-of-distribution prompts. Specifically, we consider prompts that deviate from those encountered during training, either (i) by sampling inputs from a different distribution or (ii) by introducing a mismatch between in-context examples and the query input. We describe each such prompt structure below and present a subset of the results in Figure 4 (see Appendix B.2 for additional details and the full results). In general, the model demonstrates a high degree of robustness, indicating that it has indeed learnt to perform linear regression with reasonably generality.

Recall that we construct a random training prompt $x_{\text{prompt}} = (x_1, w^T x_1, \ldots, x_k, w^T x_k, x_{\text{query}})$ by drawing the prompt inputs x_i and x_{query}, and the weight vector (w) i.i.d. from $N(0, I_d)$, with $d = 20$.

Low-dimensional subspace. We sample prompt inputs from a random 10 dimensional subspace. Crucially, unlike the training prompts, when the number of in-context examples k is between 10 and 20, the prompt inputs are linearly dependent. Nevertheless, the model is still able to achieve low error after 10 in-context examples, matching the behavior of the optimal least squares estimator.

Skewed covariance. We sample prompt inputs from $N(0, \Sigma)$ where Σ is a skewed covariance matrix with eigenbasis chosen uniformly at random and i^{th} eigenvalue proportional to $1/i^2$. We observe (see Figure 4a) that the model matches the performance of least squares until $k = 10$, mimicking the sharp drop in the error in this regime, but its error plateaus afterwards. Thus, the model is not perfectly robust here but does reasonably well, significantly outperforming the nearest neighbor baseline.



(a) Skewed covariance (b) Noisy linear regression (c) Different quadrants

Figure 4: *In-context learning on out-of-distribution prompts.* We evaluate the trained model on prompts that deviate from those seen during training by: (a) sampling from a non-isotropic Gaussian, (b) adding label noise to in-context examples, (c) restricting in-context examples to a single (random) quadrant. In all cases, the model error degrades gracefully and closely tracks that of the least squares estimator. This indicates that in-context learning generalizes beyond the training distribution.

Noisy linear regression. We add noise to each prompt output, that is, the ith output is equal to $w^T x_i + \epsilon_i$ where $\epsilon_i \sim N(0,1)$. The model matches the performance of least squares almost everywhere, and significantly outperforms it when the number of in-context examples are equal to the dimension (see Figure 4b). Interestingly, our model also exhibits the double descent curve [1] that is known to manifest for the least squares estimator [30]. Note that in this noisy setting, the optimal estimator corresponds to solving least squares with appropriate ℓ_2-regularization, however, since the model was trained on noiseless data, we cannot expect it to learn this.

Prompt scale. We consider scaling either the prompt inputs or the weight vectors, by a factor $\{1/3, 1/2, 2, 3\}$. We find that the model is quite robust when scaling the weight vector, but not as robust when scaling the prompt inputs, especially for the more extreme scales $1/3$ and 3.

Different orthants for in-context and query inputs. We fix the sign of each coordinate to be positive or negative for all in-context inputs x_i (at random). As a result, all in-context inputs lie in the same orthant, while the query input lies in another orthant with high probability. We find that the model is not affected by this mismatch and closely matches the performance of least squares (see Figure 4c). This indicates that the model is not relying on any variants of nearest neighbor search as in that case, its error would have been significantly larger (see 3-nearest neighbor baseline).

Query input orthogonal to in-context inputs. We choose the query input randomly in the space orthogonal to the space spanned by in-context example inputs. In this case, there is no information relevant to the query input in the in-context examples and thus the model correctly predicts something close to 0 to minimize the error.

Query input matches an in-context example. We choose the query input to match one of the in-context example inputs at random. Here the model predicts perfectly and is unaffected by the additional in-context examples present.

5 Beyond linear functions

We now turn our attention to in-context learning for more complex functions, namely sparse linear functions and two-layer ReLU neural networks. The overall methodology remains the same: we sample random functions from these families and train a Transformer from scratch to approximate these functions given in-context examples. (See Appendix A.3 for baseline details.)

Sparse linear functions. First, we consider functions of the form $f(x) = w^\top x$ where $w \in \mathbb{R}^d$ and has at most s non-zero coordinates. To sample a prompt $x_{\text{prompt}} = (x_1, f(x_1), \ldots, x_k, f(x_k), x_{\text{query}})$, we draw prompt inputs x_i and x_{query}, and a weight vector w from $N(0, I_d)$, and then zero out all but s coordinates of w uniformly at random. We set $d = 20$ and $s = 3$. In this setting, the least squares estimator is no longer optimal—one can perform better by leveraging the weight vector sparsity. One estimator that leverages sparsity is Lasso [45], which involves solving the least squares objective with an ℓ_1-norm regularizer for the weight vector. We plot the performance of our model in Figure 5a, and observe that our model is also able to leverage sparsity, nearly matching the performance of Lasso.

Note that this is surprising as, unlike least squares, the Lasso estimator does not have a closed form expression and hence is more challenging to compute—involving solving the regularized objective with some iterative method—while the Transformer is able to compute it in a single forward pass.

(a) Sparse linear functions (b) 2-layer neural networks (c) Linear functions

Figure 5: *Training Transformers to in-context learn more complex function classes.* (a) Transformers trained on prompts from sparse linear functions can in-context learn this class, with error decreasing at a rate similar to Lasso, and significantly better than minimum norm least squares. (b) Transformer model trained on prompts generated using random 2-layer ReLU neural networks can in-context learn this class. The error decreases at a rate similar to the baseline which involves training a neural network using gradient descent with in-context examples as the training data. (c) The same model (from (b)) can in-context learn the class of linear functions. The error decreases at a rate slower than least squares, but better that a neural network trained using gradient descent. In all cases, the errors are normalized so that the trivial zero estimator achieves an error of 1 (dashed line).

Two-layer ReLU neural networks. Next, we consider the class of two layer ReLU neural networks containing functions of the form $f(x) = \sum_{i=1}^{r} \alpha_i \; \sigma(w_i^\top x)$, where $\alpha_i \in \mathbb{R}$, $w_i \in \mathbb{R}^d$ and $\sigma(\cdot) = \max(0, \cdot)$ is the ReLU activation function. To draw a random prompt $x_{\text{prompt}} = (x_1, f(x_1), \ldots, x_k, f(x_k), x_{\text{query}})$, we sample prompt inputs x_is and x_{query} from $N(0, I_d)$, along with network parameters a_is and w_is from $N(0, 2/r)$ and $N(0, I_d)$ respectively. We set $d = 20$ and $r = 100$. In Figure 5b, we show that the model can in-context learn this class of functions from in-context samples. In fact, the error of our model matches closely the performance of the baseline which involves training a neural network performing gradient descent on in-context examples.

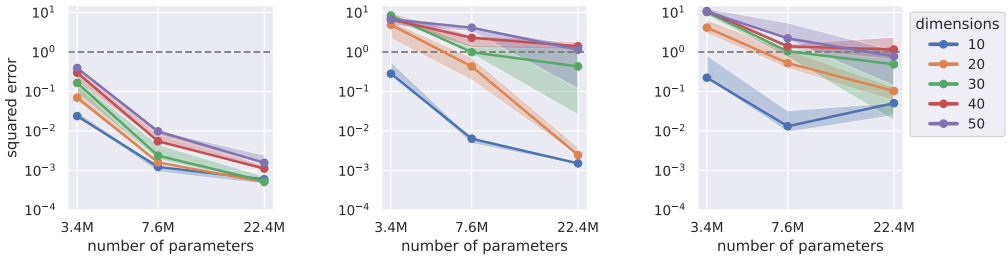
Moreover, the model trained to in-context learn two-layer neural networks is also able to in-context learn the class of linear functions (see Figure 5c), albeit with examples than least squares. This phenomenon is quite intriguing, as it demonstrates that models trained to in-context learn a rich enough function class can also in-context learn other function classes, not encountered during training.

6 Investigating what matters for in-context learning

We will now focus again on the setting of training models to in-context learn linear functions and explore the different factors that lead to successful in-context learning. Specifically, we will explore the relationship between model capacity and problem dimensionality, the impact of curriculum training, and the number of distinct functions that the model needs to be trained on to generalize.

Problem Dimension and Capacity. In Section 3 and 4, we saw that Transformer models can be trained to in-context learn 20-dimensional linear functions accurately and relatively robustly. To explore the interplay between problem dimensionality and capacity, we also consider models with fewer parameters (cf. Appendix A.1) and train each model for {10, 30, 40, 50}-dimensional problems. In Figure 6, we plot the model error with $2d$ in-context examples for various problem dimensions d as we vary the model capacity. In the standard setting, i.e., without any distribution shift, we observe (see Figure 6a) that the error decreases for each problem dimension as we increase the capacity. With fixed capacity, the error is lower for lower dimensional problems. Thus, model capacity helps perform accurate in-context learning. For out-of-distribution prompts, we observe that the the setting with skewed covariance and with in-context examples and query lying in different orthants are particularly challenging, especially for higher dimensional problems. However, we observe (see Figure 6b and

6c) that in these settings, the error decreases sharply as we increase the model capacity, even when the standard error stays roughly the same. See Appendix B.3 for additional plots.



(a) Standard (b) Different orthants (c) Skewed covariance

Figure 6: *Understanding the effect of model capacity and problem dimension on in-context learning performance for in-distribution (a) and out-of-distribution (b,c) prompts.* We train Transformers to in-context learn linear functions and plot the error with $2d$ in-context examples as we vary problem dimension d and model capacity. Capacity helps with in-context learning in most cases, especially on out-of-distribution prompts (even when the gains in the standard setting are small). We train 3 models in each case with different random seeds, and show the median error (solid lines), and the minimum and maximum errors (shaded region). (See Appendix B.4 for training variance analysis.)

Curriculum. We use curriculum learning drawing the prompt inputs from a low-dimensional subspace initially (by setting some of the coordinates to 0), and gradually increasing the subspace dimension. Equivalently, this can be viewed as gradually increasing the complexity of the function class. This speeds up training drastically, especially for higher dimensional problems: for dimension 40 and 50, the squared error barely decreases through the 500k training steps without curriculum but goes to almost 0 with curriculum. For the 20 dimensional problem where we were able to train the model without curriculum within the training (step count) budget, we did not observe any qualitative difference in accuracy or robustness compared to the model trained with curriculum. We include plots comparing the speed and accuracy of training with and without curriculum in Appendix B.5.

Interestingly, there is an initial relatively long period in training where the loss does not decrease, followed by a period of sharp decrease. The length of this period increases with problem dimension, and yet is pronounced in curriculum training. Understanding the model just before and after this moment is a promising future direction, which can give insights into the emergence in-context learning. Intriguingly, Olsson et al. [31] observe a similar sudden improvement in the in-context learning ability of a language model which they attribute to the formation of "induction heads".

Number of distinct functions seen during training. In order to understand the training data required for in-context learning, we perform an ablation where we limit the number of distinct functions used for training. That is we create a set of n_w randomly chosen vectors and sample weight vectors uniformly from that set to generate the training prompts (the inputs are still sampled at random for each training prompt). We observe that the model demonstrates non-trivial performance with $n_w = 100$ and close to optimal performance with $n_w = 10000$ (see Appendix B.6 for details) For context, in Section 3, the model is trained on a new random prompt at each step and thus encounters 32M distinct linear functions (500k training steps with a batch size of 64).

7 Related work

In-context learning. Since Brown et al. [7] demonstrated the in-context learning ability of GPT-3, there has been a significant interest in improving this ability [20, 23, 51, 22, 40, 24, 9, 27, 17], as well as understanding the mechanism behind it. The works most relevant to ours are as follows. Xie et al. [48] propose a Bayesian inference framework explaining how in-context learning works despite formatting differences between training and inference distributions. Razeghi et al. [38] show that in-context learning for numerical reasoning tasks is better for instances whose terms are more prevalent in training data. Min et al. [23] demonstrate tasks where in-context learning works even when the prompt outputs are chosen randomly, questioning to what extent these models are truly learning new tasks on-the-fly, while Rong [39] gives examples of novel tasks on which these models

demonstrate on-the-fly learning ability. Chan et al. [8] demonstrate that distributional properties such as long-tailedness are crucial for in-context learning on an image-based few-shot dataset. Olsson et al. [31] and Elhage et al. [12] hypothesize the existence of special circuits inside transformer models responsible for in-context learning, that can complete prompts by copying previous similar patterns in the prompt sequence. Our work contributes to and complements this line of work on understanding in-context learning, by posing it as a well-defined problem of learning function classes at inference time, and empirically investigating the in-context learnability of fundamental function classes.

Transformers. There is a long line of work investigating the capabilities [46, 10, 50, 33, 49, 5], limitations [15, 4] and applications [21, 11, 32] of Transformers. Most similar to our work, Müller et al. [28] introduce a "Prior-data fitted transformer network" that is trained to approximate Bayesian inference and generate predictions for downstream learning problems. However, while their focus is on being able to do Bayesian inference faster than existing methods such as MCMC and using their network for downstream applications (with or without parameter fine-tuning), our work focuses on formalizing and understanding in-context learning by considering fundamental function classes.

Meta learning. In-context learning can be viewed as an instance of the more general learning-to-learn or meta-learning paradigm [44, 29]. Typical approaches from this extensive line of work (see [16] for a survey) include: training a meta-learner on how to update the parameters of a downstream learner [2, 18], learning parameter initializations from which one can quickly train for many downstream tasks [14, 37], learning latent embeddings that allow for effective similarity search [43]. Most relevant to our setting are approaches that directly take as input examples of a downstream task and a query input and produce the corresponding output Mishra et al. [26], Santoro et al. [42]. These approaches typically focus on real-world datasets (e.g., images) and thus are complementary to our work.

8 Discussion

In this work, we formalize and study the question: what classes of functions can a model learn in-context? We show that a Transformer model trained from scratch can in-context learn the class of linear functions, with performance comparable to the optimal least squares estimator, even under distribution shifts. Moreover, we show that in-context learning is possible for the class of sparse linear functions and two-layer neural networks; learning problems which are solved in practice with involved iterative algorithms such as gradient descent. Moreover, the model trained to in-context learn two-layer neural networks is also able to in-context learn linear functions. Thus, not only can Transformers in-context learn unseen functions from the class they have been trained on, but can also generalize to other function classes. Our work lays the groundwork for several future directions.

Complexity of in-context learning. We empirically show that model capacity helps in performing in-context learning accurately and robustly. This raises the question: How does the in-context learning loss (1) depend on the complexity of the function class $\mathcal{F}$, the capacity of model M, and the number of in-context examples used to train M. Even the right notion of complexity of $\mathcal{F}$ is unclear and may depend on the model family. Understanding this question for models explicitly trained to perform in-context learning may suggest lower bounds for the in-context learning performance of models such as GPT-3 that have not been explicitly trained for this purpose.

Curriculum training. Within our framework, there is natural notion of curriculum training where we gradually increase the complexity of the function class which we in-context learn. This leads to drastic speed-ups in training. What is the reason behind such a speedup? Are similar speedups also possible for training large language models? Understanding this question can have implications for training of models on real datasets, potentially reducing in the time and energy used for training.

Inductive bias of model families. Language models are trained on raw data scraped from the Internet which makes it difficult to understand their inductive bias. Our framework presents an opportunity to understand and compare the inductive biases of different model families (e.g., Transformers vs. LSTMs) in a well-defined setting. For instance, a concrete question is: Are their function classes that are easier to in-context learn using transformers but harder for LSTMs and vice-versa?

Algorithm discovery. For settings such as sparse linear regression or 2-layer neural networks, we do not always have a good understanding of what the optimal algorithms are. Nevertheless, in Section 5 we found that Transformers are able to discover such algorithms when being training to perform in-context learning. This suggest an intriguing possibility where we might be able to reverse engineer the learning algorithm encoded in the Transformer to obtain new algorithms for such problems.

References

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 2019.

[2] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, 1995.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, pages 41–48, 2009.

[4] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

[5] Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. *arXiv preprint arXiv:2006.09286*, 2020.

[6] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Neural Information Processing Systems (NeurIPS)*, 2020.

[8] Stephanie CY Chan, Adam Santoro, Andrew K Lampinen, Jane X Wang, Aaditya Singh, Pierre H Richemond, Jay McClelland, and Felix Hill. Data distributional properties drive emergent few-shot learning in transformers. *arXiv preprint arXiv:2205.05055*, 2022.

[9] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. Meta-learning via language model in-context tuning. *arXiv preprint arXiv:2110.07814*, 2021.

[10] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[13] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 1993.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning (ICML)*, 2017.

[15] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 2020.

[16] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[17] Andrew K Lampinen, Ishita Dasgupta, Stephanie CY Chan, Kory Matthewson, Michael Henry Tessler, Antonia Creswell, James L McClelland, Jane X Wang, and Felix Hill. Can language models learn from explanations in context? *arXiv preprint arXiv:2204.02329*, 2022.

[18] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[19] Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 2021.

[20] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[21] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

[22] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

[23] Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Noisy channel language model prompting for few-shot text classification. *arXiv preprint arXiv:2108.04106*, 2021.

[24] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

[25] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[26] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations (ICLR)*, 2018.

[27] Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. Reframing instructional prompts to gptk's language. *arXiv preprint arXiv:2109.07830*, 2021.

[28] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

[29] Devang K Naik and Richard J Mammone. Meta-neural networks that learn by learning. In *International Joint Conference on Neural Networks (IJCNN)*, 1992.

[30] Preetum Nakkiran. More data can hurt for linear regression: Sample-wise double descent. *arXiv preprint arXiv:1912.07242*, 2019.

[31] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

[32] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning (ICML)*, 2018.

[33] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

[34] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[36] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[37] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[38] Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

[39] Frieda Rong. Extrapolating to unnatural language processing with gpt-3's in-context learning: The good, the bad, and the mysterious), 2021. URL `http://ai.stanford.edu/blog/in-context-learning/`.

[40] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*, 2021.

[41] Terence D Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE transactions on Robotics and Automation*, 1994.

[42] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning (ICML)*, 2016.

[43] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Neural Information Processing Systems (NeurIPS)*, 2017.

[44] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[45] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1996.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Neural Information Processing Systems (NeurIPS)*, 2017.

[47] Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work? *arXiv preprint arXiv:2012.03107*, 2020.

[48] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations (ICLR)*, 2022.

[49] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*, 2021.

[50] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

[51] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning (ICML)*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We don't perceive any negative impacts beyond those of building large-scale models in general.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Appendix A.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix A.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]