Blog Post Track

ICLR

Understanding in-context learning in transformers

Simone Rossi, Rui Yuan, Thomas Hannagan
Stellantis (France)

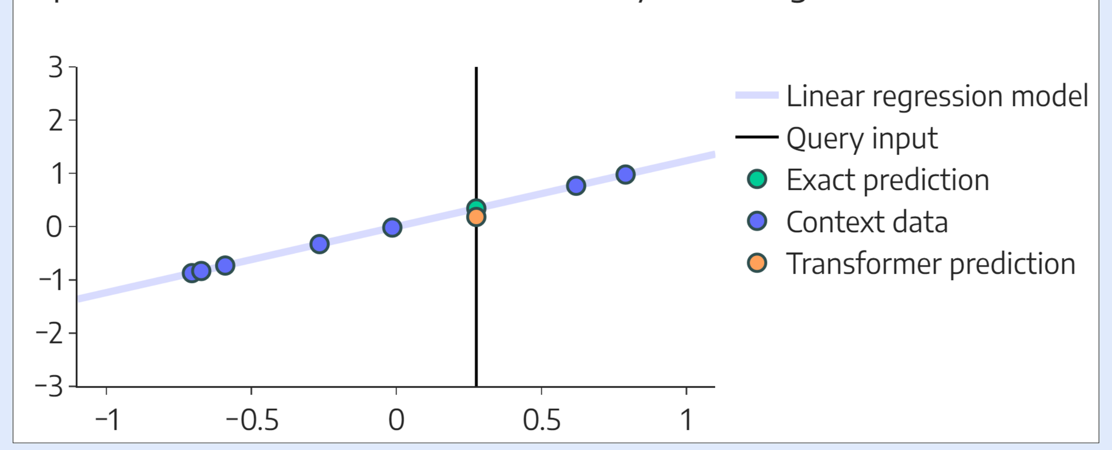
STELLANTIS

Blog Post: Objective and Contributions

Informal definition: In-Context Learning (ICL) is a behavior observed in Large Language Models (LLMs), where learning occurs from prompts using unmodified model weights.

Question: How can we explain the behavior of ICL in LLMs?

Simplification: We need to simplify the problem. Our focus will be on linear regression tasks using linear self-attention models with a single head.



Main reference for this blog post

Transformers Learn In-Context by Gradient Descent by Oswald et al. [1].

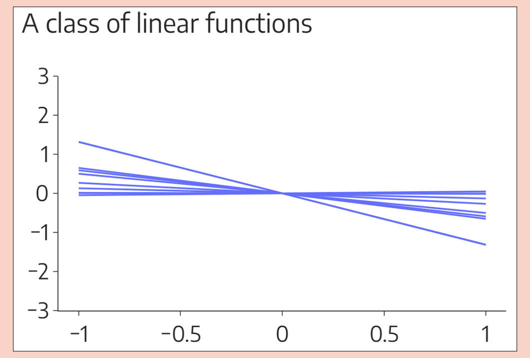
How:

1. Demonstrate and explain how and why ICL occurs in transformer architectures.
2. Analyze ICL through the lens of optimization theory.
3. Discuss the theoretical framework in [1] to show the equivalence between ICL and gradient descent.

In-Context Learning for Linear Regression

Formalizing in-context learning

A model demonstrates in-context learning for a function class $\mathcal{H}$ if, for any function $h \in \mathcal{H}$, it can approximate $h(x_{query})$ for any new input x_{query} using unseen in-context examples $\{(x_i, y_i)\}_{i=0}^{C-1}$, where $y_i = h(x_i)$, and C is the fixed context size, without modification of the model's weights.



Goal: To study ICL for linear regression tasks of the form $h_w(x) = w^\top x$ from a dataset of **unseen in-context examples** $\{(x_i, y_i)\}_{i=0}^{C-1}$, with $x_i, w \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$.

Plan:

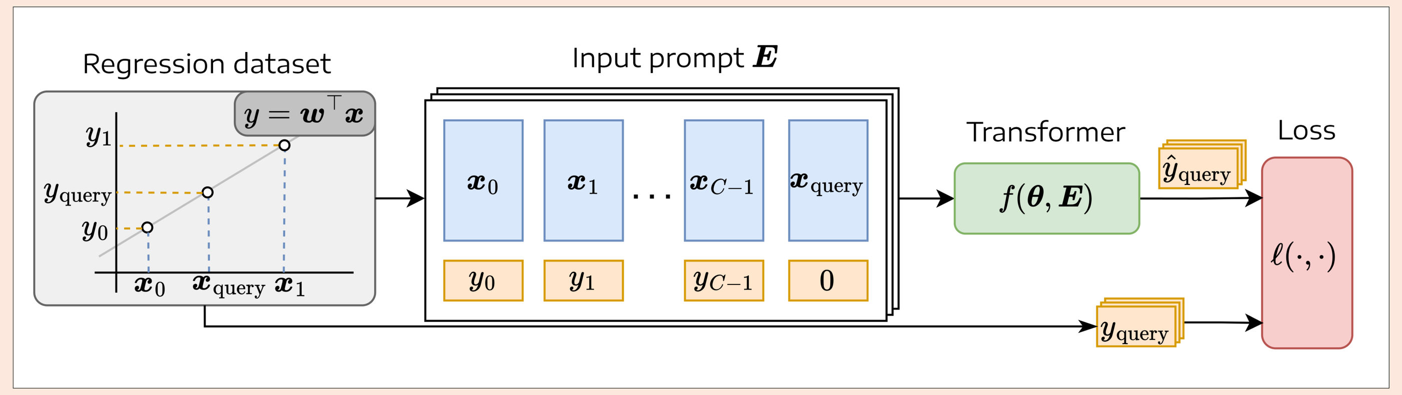
- Under what conditions can a transformer learn in-context?
- How does a transformer learn in-context under these conditions?

References

[1] J. von Oswald et al. "Transformers Learn In-Context by Gradient Descent". Proceedings of the 40th International Conference on Machine Learning 2023.

[2] S. Biderman et al. "Emergent and Predictable Memorization in Large Language Models". Thirty-seventh Conference on Neural Information Processing Systems 2023.

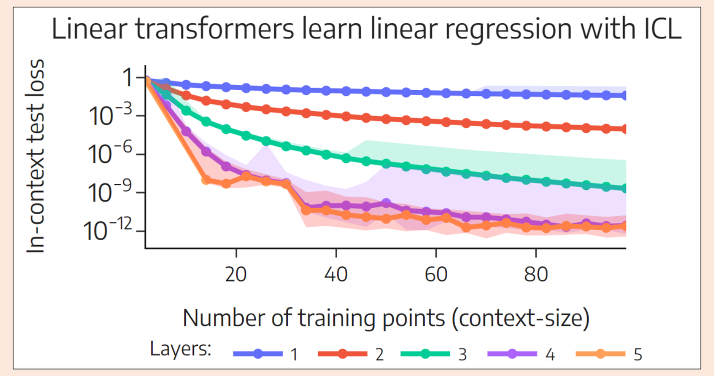
Pre-training Setup



- **Dataset construction**: We sample $w \sim p(w)$ and C inputs $x_i \sim p(x)$, where C is the fixed context size. We then compute $y_i = w^\top x_i$ and prepare the input sequence E.
- **Model definition**: We use **linear transformer** which replace softmax self-attention with linear self-attention, $f(\theta, E) = E + W_P W_V E (W_K E)^\top W_Q E$.
- **Pre-training**: We optimize the model f by minimizing the regression loss on the query point x_{query} with context $\{(x_i, y_i)\}_{i=0}^{C-1}$

$$\mathcal{L}(\theta) = \mathbb{E}_{w,x} \left\| f\left(\theta, \left[\{x_i, y_i\}_{i=0}^{C-1}, x_{query}\right]\right) - y_{query} \right\|^2$$

Linear transformers can learn linear functions in-context well. The test loss decreases as the context size increases, and as the number of layers increases.



Connection Between In-Context Learning and Gradient Descent

Observation: A gradient descent update is a linear transformation of the data.

- Write gradient descent update of the least squares loss

$$\mathcal{L}_{lin}\left(w, \{x_i, y_i\}_{i=0}^{C-1}\right) = \frac{1}{2C}\sum_{i=0}^{C-1}\left(w^\top x_i - y_i\right)^2 \quad \text{and} \quad \Delta w \overset{\text{def}}{=} \eta \nabla \mathcal{L}_{lin} = \frac{\eta}{C}\sum_{i=0}^{C-1}\left(w^\top x_i - y_i\right) x_i$$

- Compute the loss after applying a gradient descent step

$$\mathcal{L}_{lin}\left(w - \Delta w, \{x_i, y_i\}_{i=0}^{C-1}\right) = \frac{1}{2C}\sum_{i=0}^{C-1}\left(w^\top x_i - y_i - \Delta w^\top x_i\right)^2$$

- Let $\hat{x}_i = x_i$ and $\hat{y}_i = y_i + \Delta w^\top x_i$, then $\mathcal{L}_{lin}\left(w - \Delta w, \{x_i, y_i\}_{i=0}^{C-1}\right) = \mathcal{L}_{lin}\left(w, \{\hat{x}_i, \hat{y}_i\}_{i=0}^{C-1}\right)$.

Important note: It shows that we can achieve the same loss as after one gradient step by adjusting the inputs and targets while keeping the weights fixed.

Now, we define a linear transformer that implements one step of gradient descent (GD) of the least squares loss with initialization w_0, which we call a **GD-transformer**:

$$W_K = W_Q = \begin{pmatrix} I_D & 0 \\ 0 & 0 \end{pmatrix} \qquad W_V = \begin{pmatrix} 0 & 0 \\ w_0^\top & -1 \end{pmatrix} \qquad W_P = \frac{\eta}{C} I_{D+1}$$

To prove this, it is sufficient to plug in the GD-transformer to obtain the output:

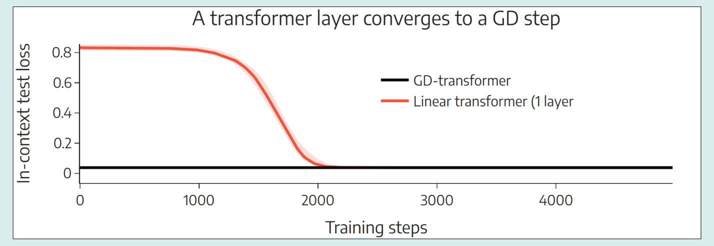
$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + W_P W_V E E^\top W_K^\top W_Q \begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} 0 \\ -\Delta w_0^\top x_j \end{pmatrix} \quad \forall j \in \{0, \dots, C-1\} \text{ and } x_{query}.$$

Conclusion: We have shown that a linear transformer can be constructed to implement GD on the least squares loss, which suggests that the GD-transformer has the ability of ICL. **But**, would a linear transformer converge to this GD-transformer after pre-training?

A linear transformer learns to implement gradient descent

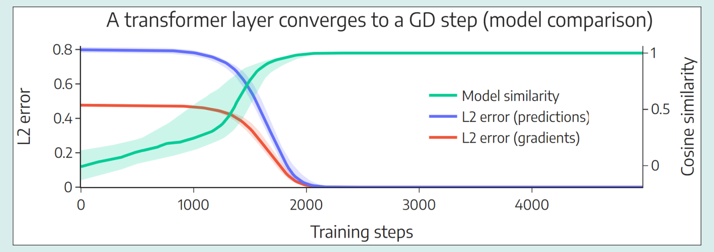
Question: During pre-training, what loss is the linear transformer optimizing in-context?

- The loss of the linear transformer converges to the loss of the GD-transformer, which, by construction, implements one step of gradient descent.



Question: During pre-training, what is a linear transformer learning to implement?

- The predictions and gradients of the linear transformer converge to those of the GD-transformer.

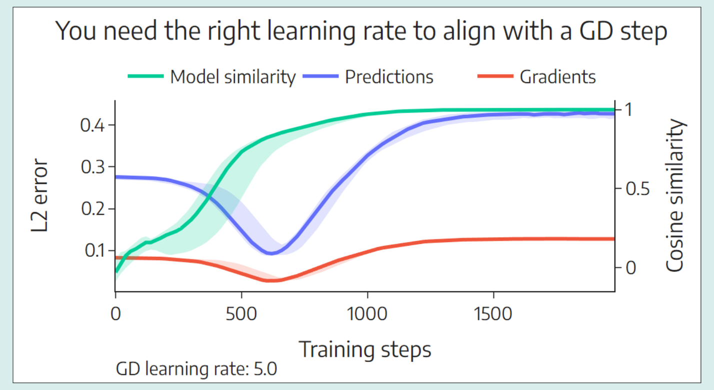


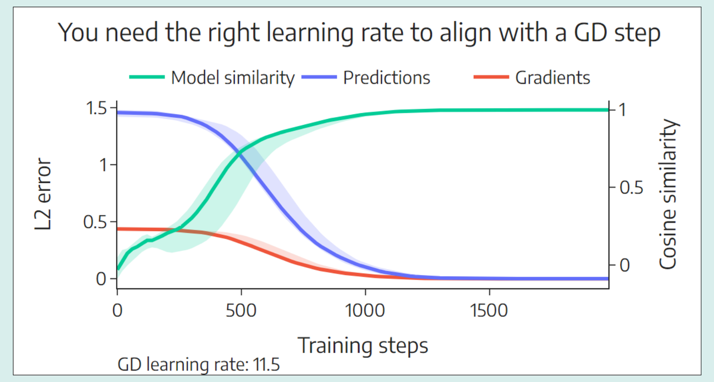
Conclusion:

Because the ground truth is the same for both models, it must mean that the models are converging to the same outputs given the same inputs, which implies that **they are implementing the same function**.

Analysis on the Learning Rate of Gradient Descent

- The GD learning rate is a key hyperparameter for the GD-transformer construction.
- The linear transformer converges to the loss of the GD-transformer **specifically** for a single value of the GD learning rate, determined through line search.





Important note: The optimal GD learning rate for the GD-transformer can be analytically derived by optimizing the quadratic function $\eta^* = \arg\min_{\eta \in \mathbb{R}} \mathcal{L}_{lin}\left(w - \Delta w, \{x_i, y_i\}_{i=0}^{C-1}\right)$.

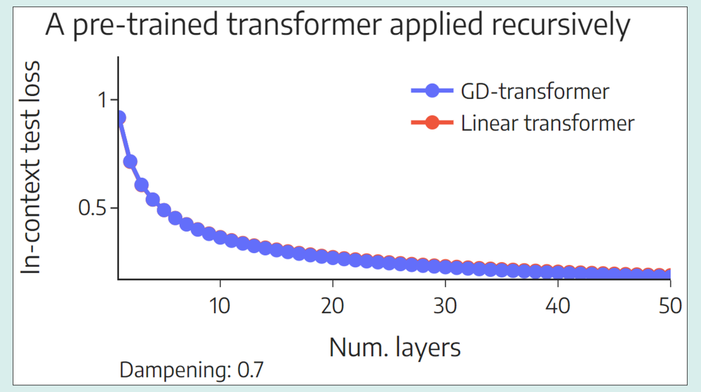
Conclusions: During pre-training, the linear transformer (1) learns to implement a GD step and (2) implicitly optimizes the GD learning rate.

How about multiple layers?

We recurrently apply the same layer with the same weights multiple times. Specifically, for the embedding matrix $E^{(l)}$ at layer l, we update the linear transformer as follows:

$$E^{(l+1)} = E^{(l)} + \lambda W_P W_V E^{(l)} \left(W_K E^{(l)}\right)^\top W_Q E^{(l)} \quad \text{with a dampening factor } \lambda \in (0, 1)$$

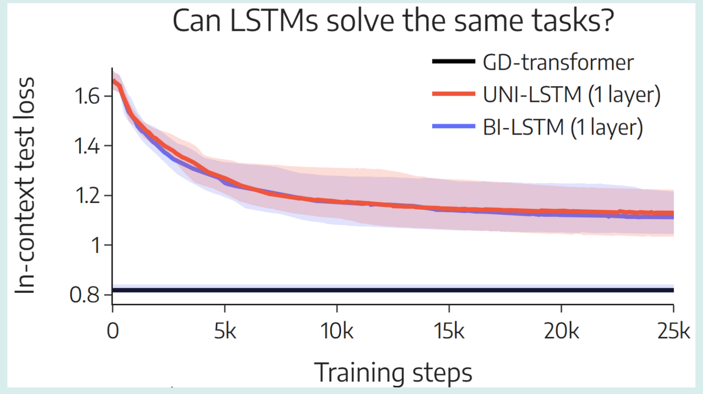
- The test losses for both the linear transformer and the GD-transformer diverge as the number of layers increases.
- The test loss of the linear transformer with multiple layers matches that of the GD-transformer with multiple layers for a specific value of the dampening factor, $\lambda = 0.7$.

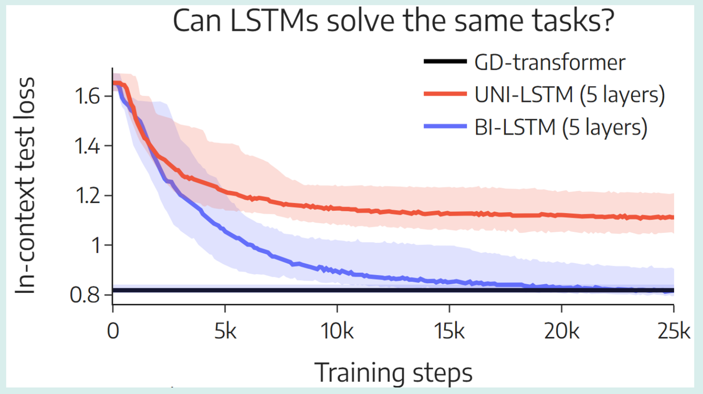


Take a picture to check the full blog post

Is this behavior unique to transformers?

We apply the same setup to investigate the behavior of uni and bidirectional LSTMs.

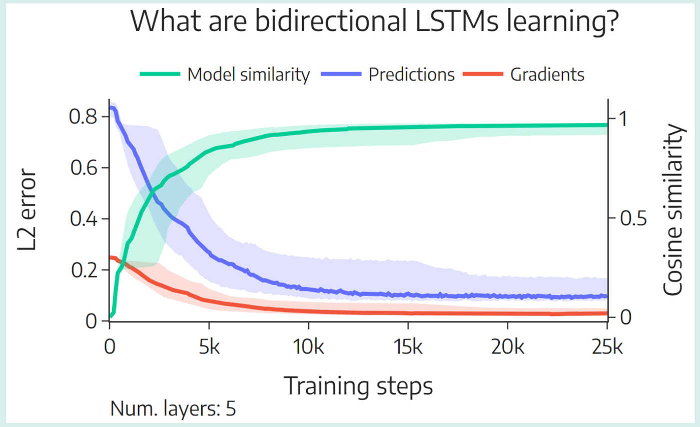




- LSTMs struggle to learn linear functions in-context as effectively as transformers.
- Bidirectional LSTMs perform better, especially with an increased number of layers.

Observations:

- Regardless of the number of layers, the unidirectional LSTM does not implement a gradient descent step.
- For two or more layers, the bidirectional LSTM behaves increasingly like a gradient descent step, though the cosine similarity does not reach 1.



Conclusions and Discussion

In this blog post,

1. We have presented the approach in [1] of the in-context learning (ICL) phenomenon and we explained how transformers can do ICL through the implementation of a gradient descent (GD) step.
   - We discussed how the GD-transformer, by construction, can execute a GD step in-context.
   - We demonstrated how, during pre-training, the transformer learns to execute a GD step.
2. We replicated and extended the findings of the original paper [1] by:
   - Providing an analytical solution for the learning rate in the GD-transformer.
   - Conducting a study on the behavior of uni and bidirectional LSTMs.
3. We discussed some limitations and highlighted potential research directions.

Open questions

- Can we identify scaling laws for ICL in large models?
- Do these results generalize to different architectures, such as state-space models (e.g., MAMBA)?
- What limitations are inherent to the optimization lens? Could emergent abilities and memorization [2] suggest alternative mechanisms?
- How can other frameworks (e.g., mesa-optimization, meta-learning) help us better understand ICL in large models?