ETH zürich

Google Research

Transformers Learn In-Context by Gradient Descent

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, Max Vladymyrov

ICML International Conference On Machine Learning

Link to the paper and code

Goal & Hypothesis

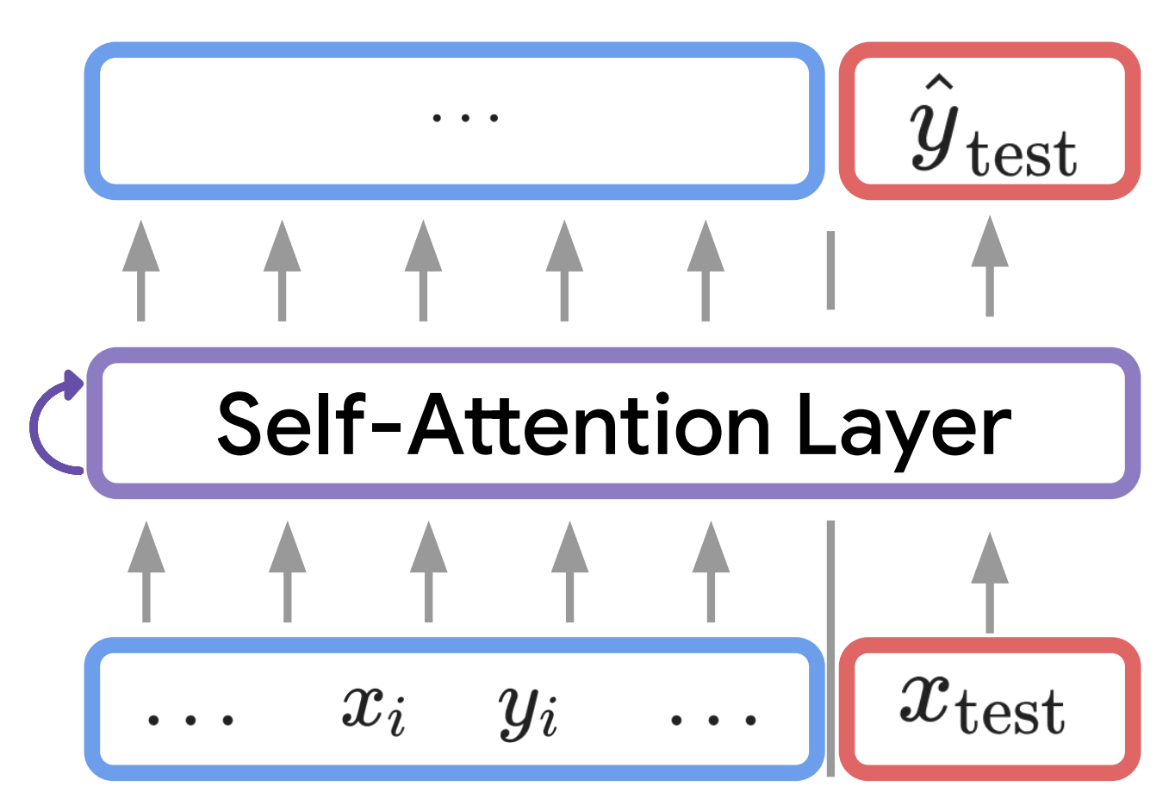
Understand better Transformers and especially their intriguing in-context learning capability.

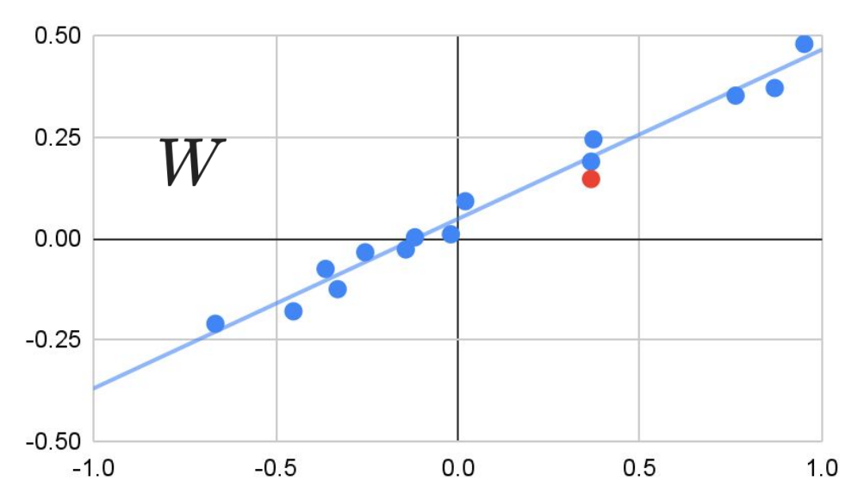
- Garg et al.[2] and Akyürek et al.[3]: trained Transformers on few-shot learning tasks often resemble gradient descent.
- Our goal: to explain this phenomenon by building on the relationship between self-attention and fast weight programming [Schmidhuber, 1992][1].

Contributions:

1) Construction of linear attention weights equivalent to do steps of GD on linear regression.
2) Evidence that this construction is found in practice
3) Show how MLPs in the architecture enables solving non-linear tasks
4) Relax assumption of construction by showing Transformers learn to copy

Setting





$$\hat{y}_{\text{test}} = t_\theta(x_{\text{test}}, \{(x_i, y_i)\}_{i=1}^N)$$
$$\text{where} \quad y_i = W x_i$$

Main Insights & Construction

Linear attention, if presented with correctly pre-processed data, can implement a step of gradient descent on the squared error regression loss.

Compare GD

1) Compute regression loss: $L(W, \{(x_i, y_i)\}_{i=1}^N) = \frac{1}{2N}\sum_{i=1}^N (Wx_i - y_i)^2$
2) Gradient descent: $\Delta W = -\eta \nabla_W L = -\frac{\eta}{N}\sum_{i=1}^N (Wx_i - y_i)x_i^T$
3) Trick: Update all y: $L(W + \Delta W, \{(x_i, y_i)\}_i^N) = L(W, \{(x_i, y_i - \Delta W x_i)\}_i^N$
4) Correct test prediction : $\hat{y}_{\text{test}} \leftarrow -1 \cdot \hat{y}_{\text{test}} = -1 \cdot \sum -\Delta W x_{\text{test}}$

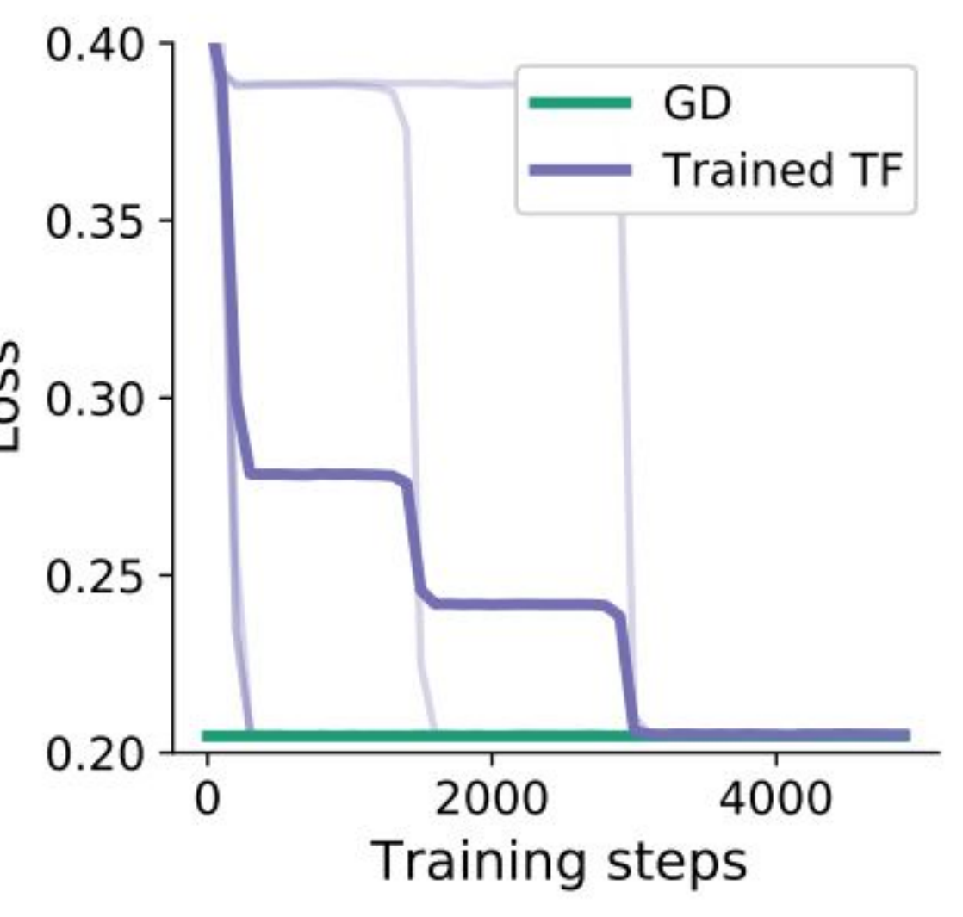
and linear Self-Attention GD

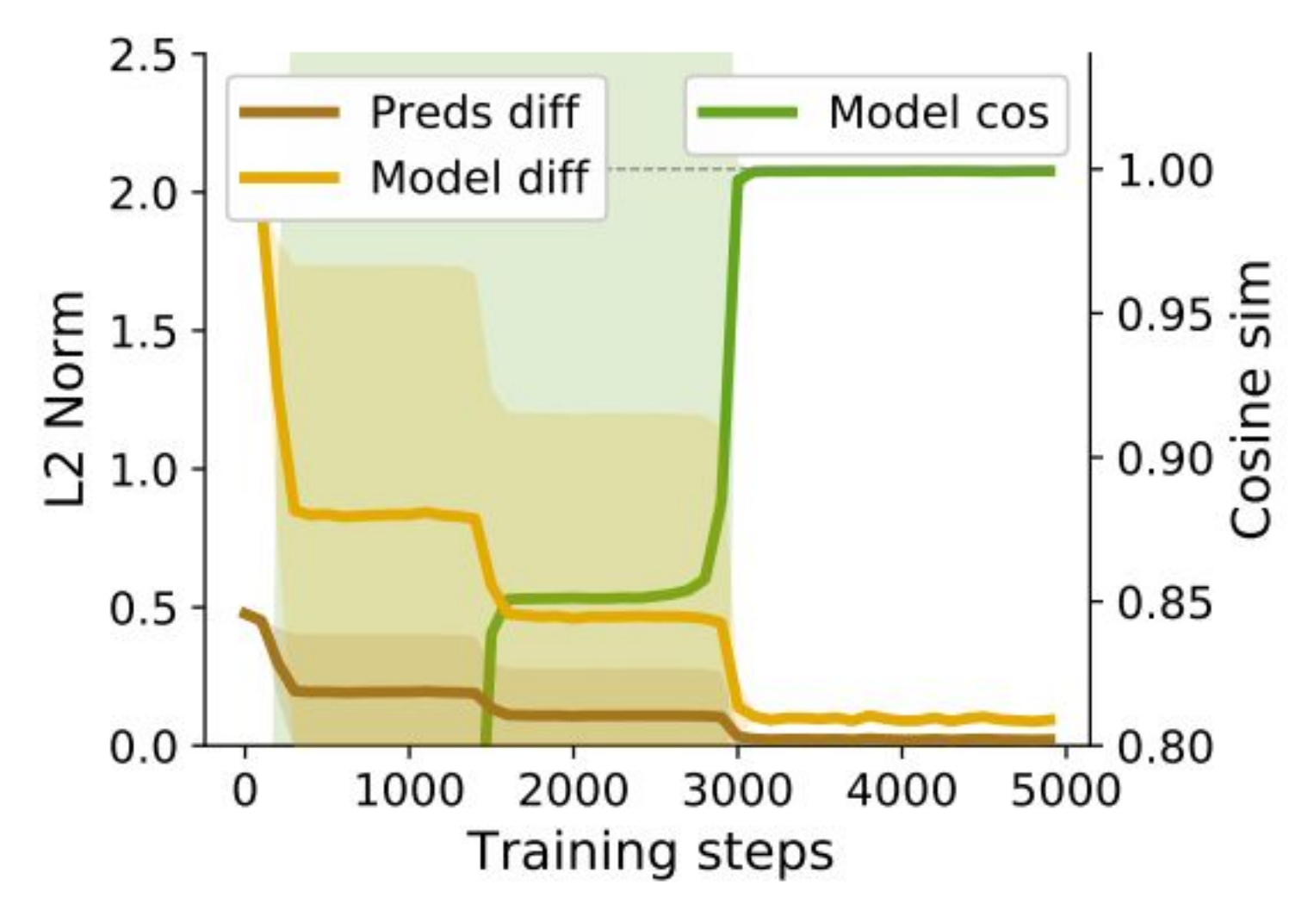
1) Assume token construction of copied data: $e_i = (x_i, y_i)$
2) Update tokens by linear self-attention: $e_i \leftarrow e_i + PVK^T q_i$
3) Can implement GD: $(x_i, y_i) \leftarrow (x_i, y_i) - (0, \frac{\eta}{N}\sum_j (Wx_j - y_j)x_j^T x_i)$

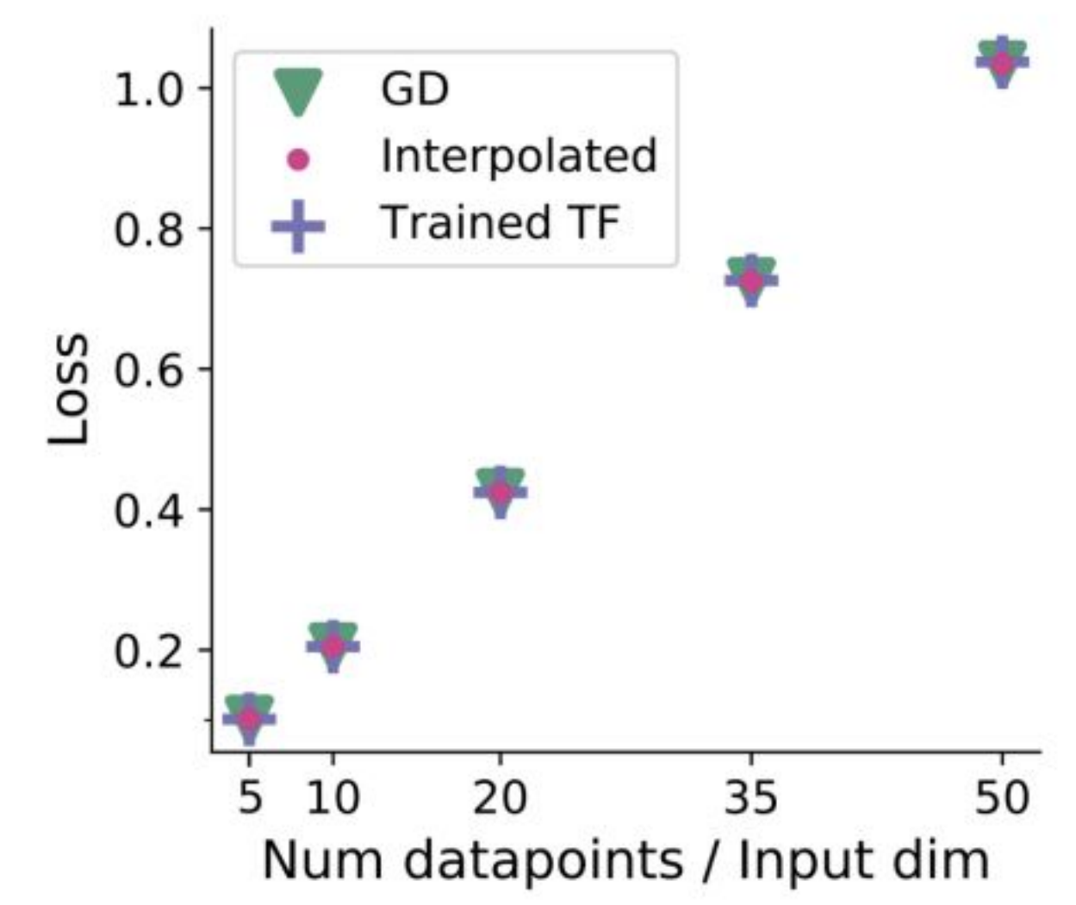
How Transformers can solve *linear* regression tasks

We present several pieces of evidence for the hypothesis that our construction is equivalent to what a trained transformer actually learns.

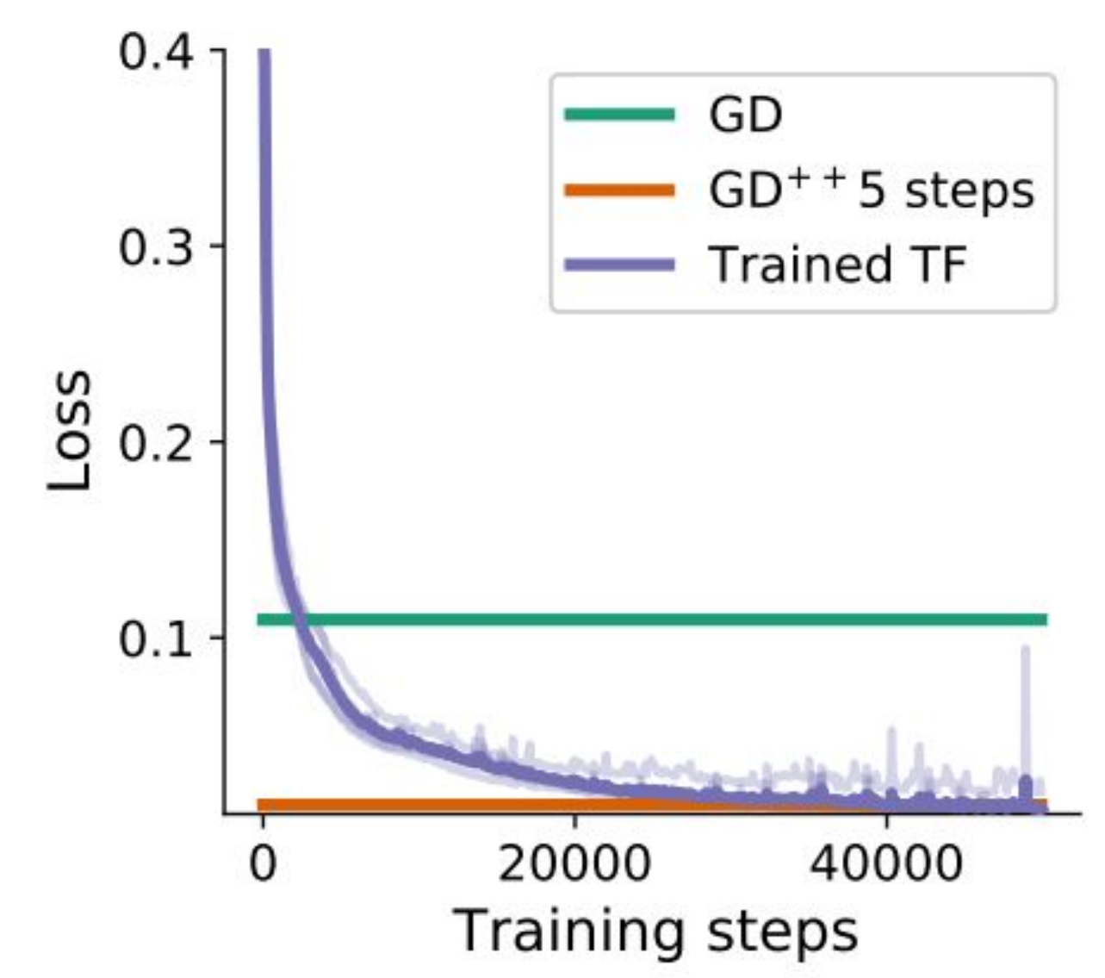
1) Trained single linear self-attention layer

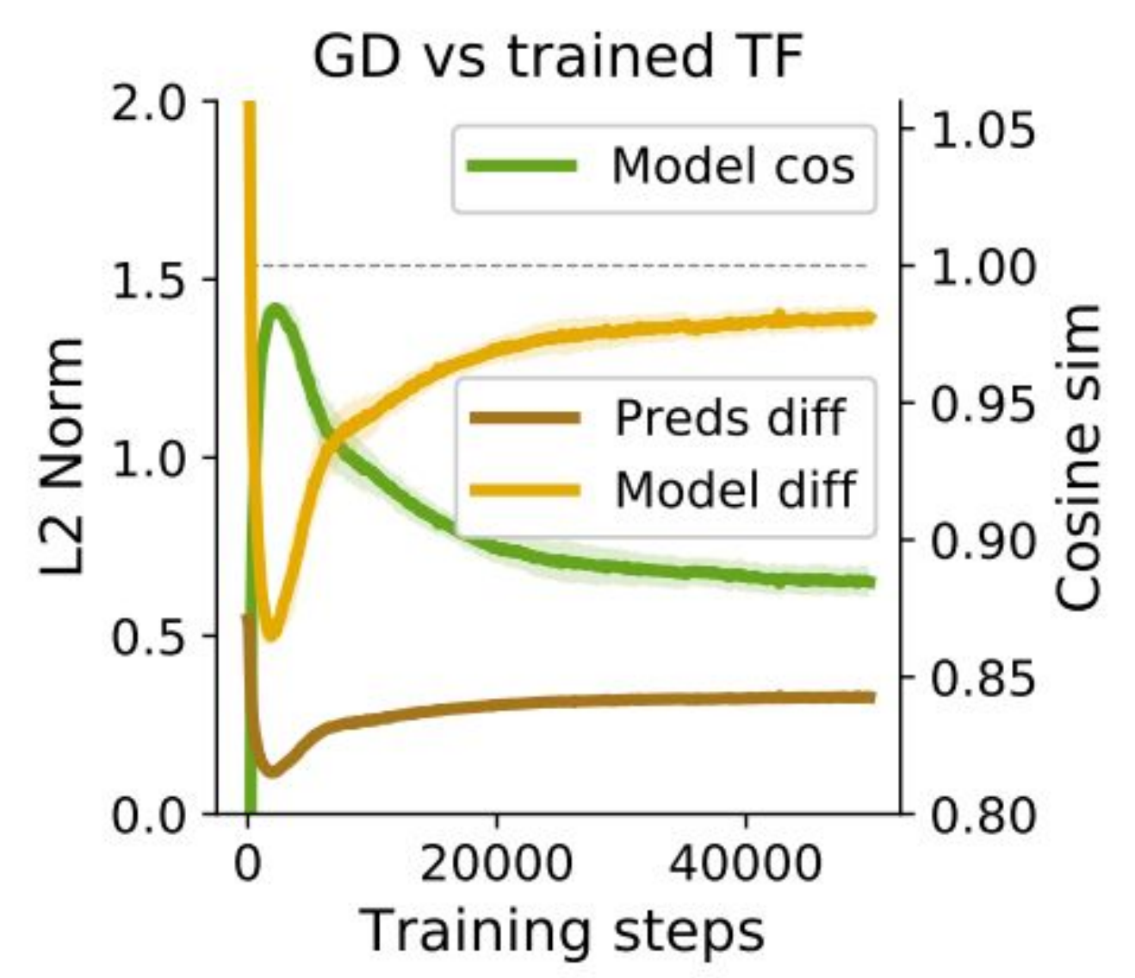


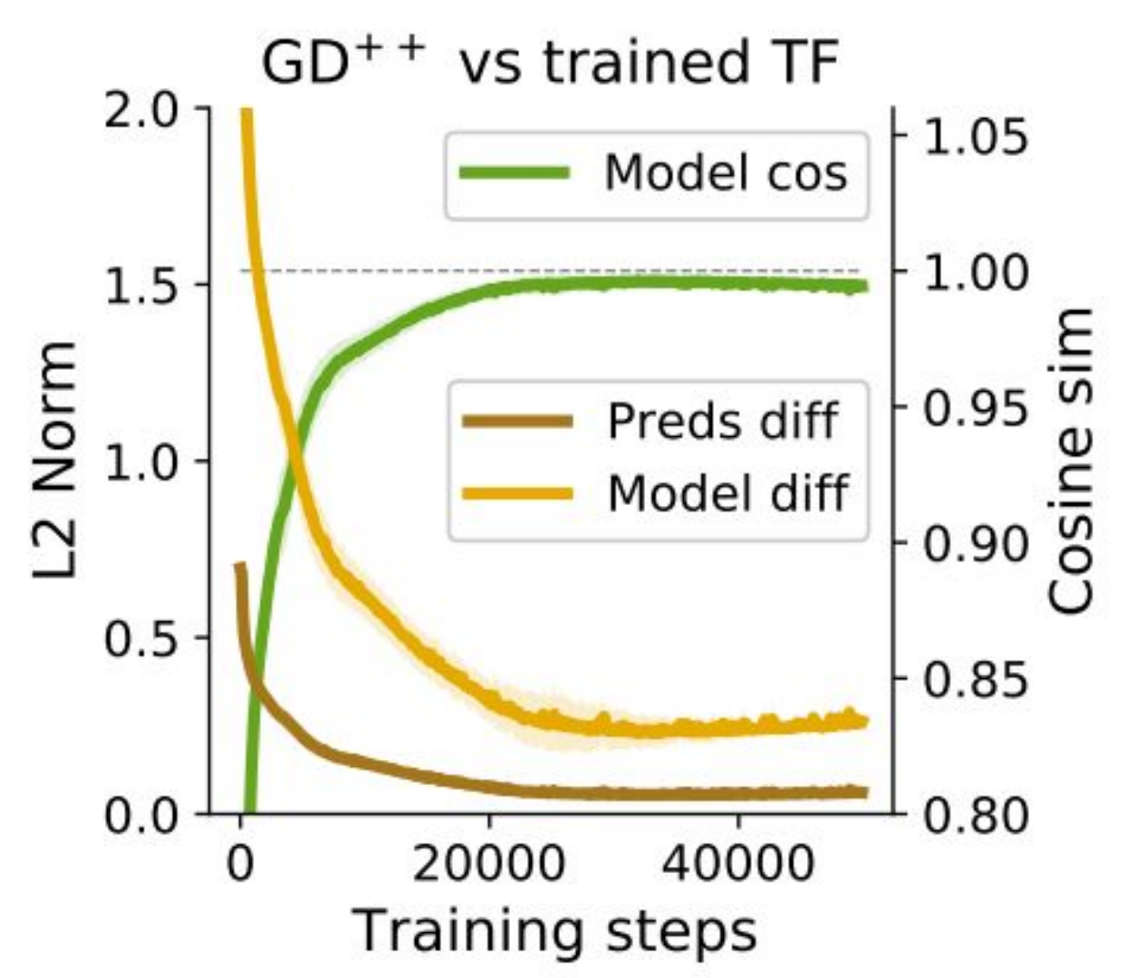




2) Trained Transformer of 5 linear self-attention layers







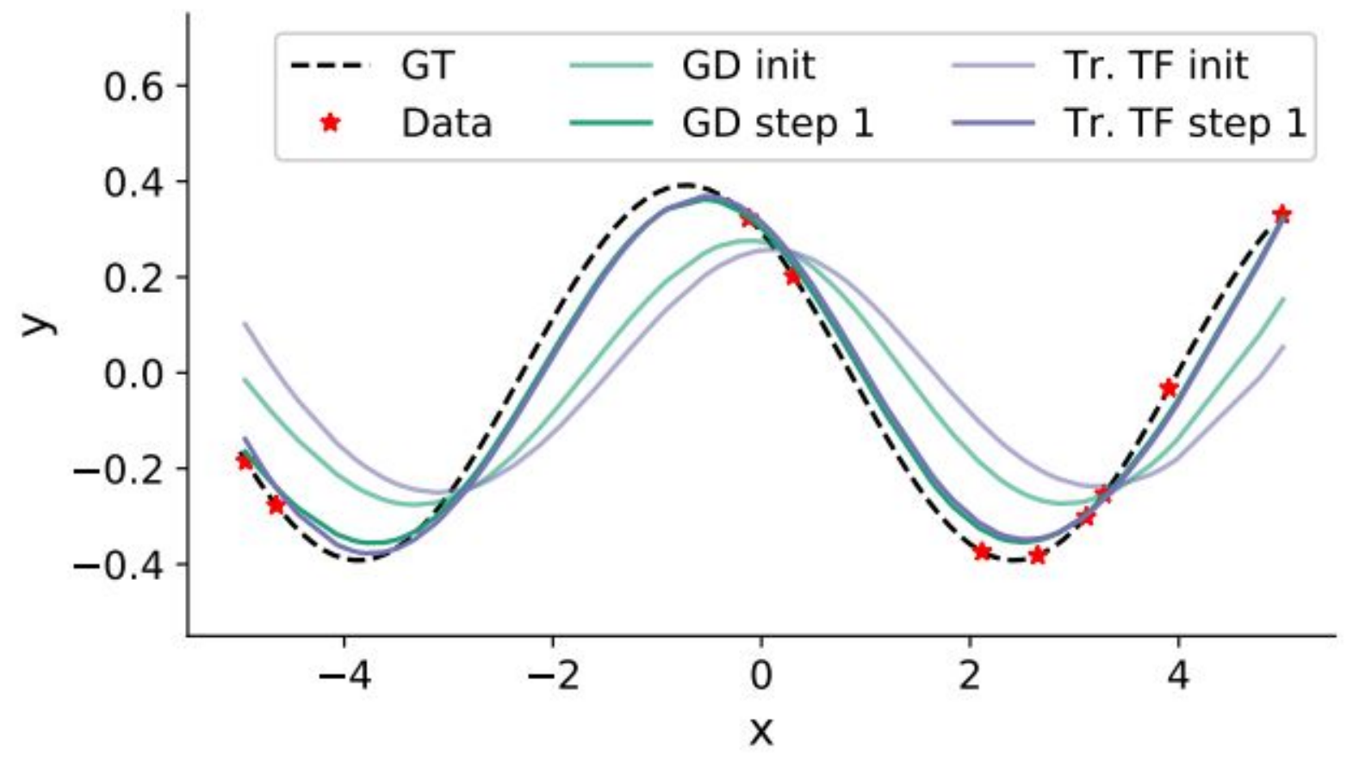
Multi layer Transformer outperform plain gradient descent and iteratively transform the input data X, as well as the targets Y with GD. We term this algorithm GD++

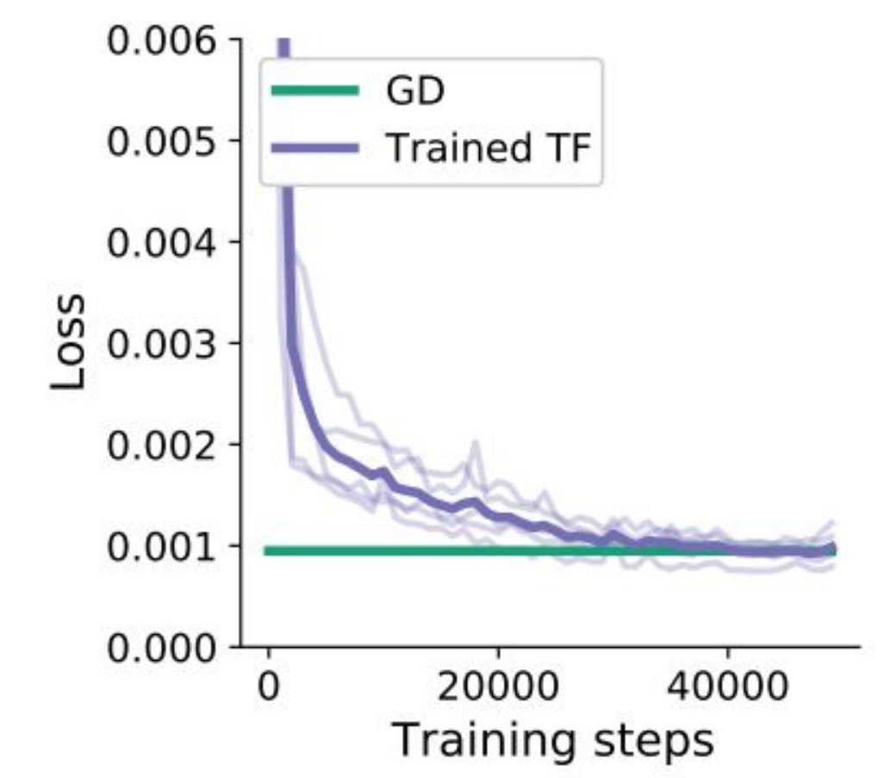
$$(x_i, y_i) \leftarrow (x_i, y_i) - (\gamma XX^T x_i, \Delta W x_i)$$

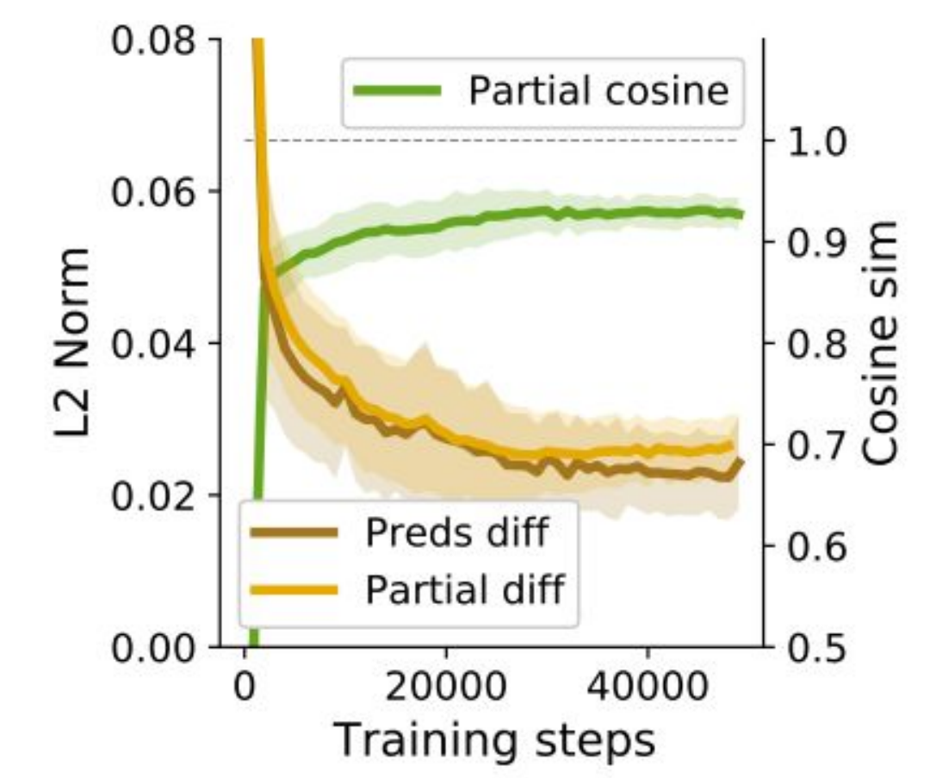
Key takeaway: When trained on linear regression taks, multi-layer linear self-attention Transformers implement GD, GD++ or *behave* very similarly.

How Transformers can solve *non-linear* regression tasks

We hypothesize and provide some evidence that Transformers exploit MLPs to non-linearly embed data and solve non-linear regression tasks by gradient descent.

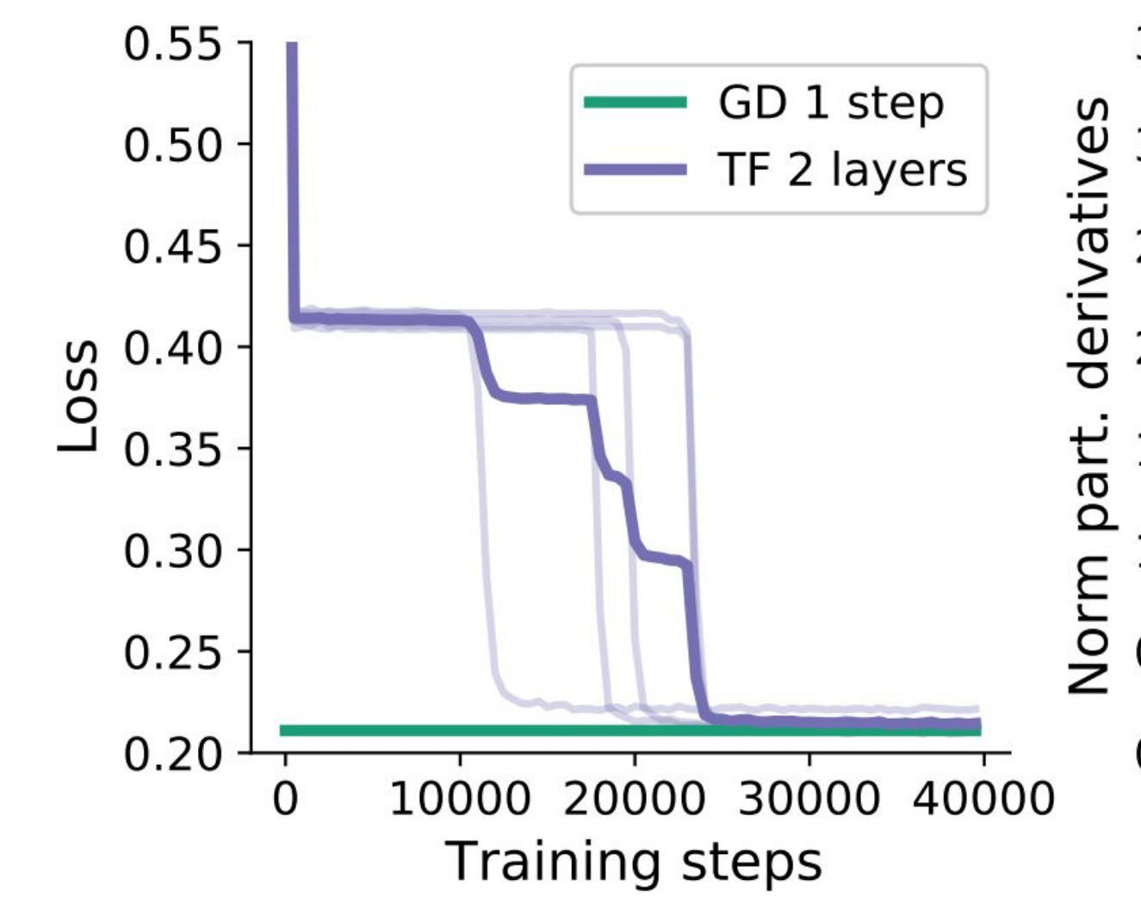


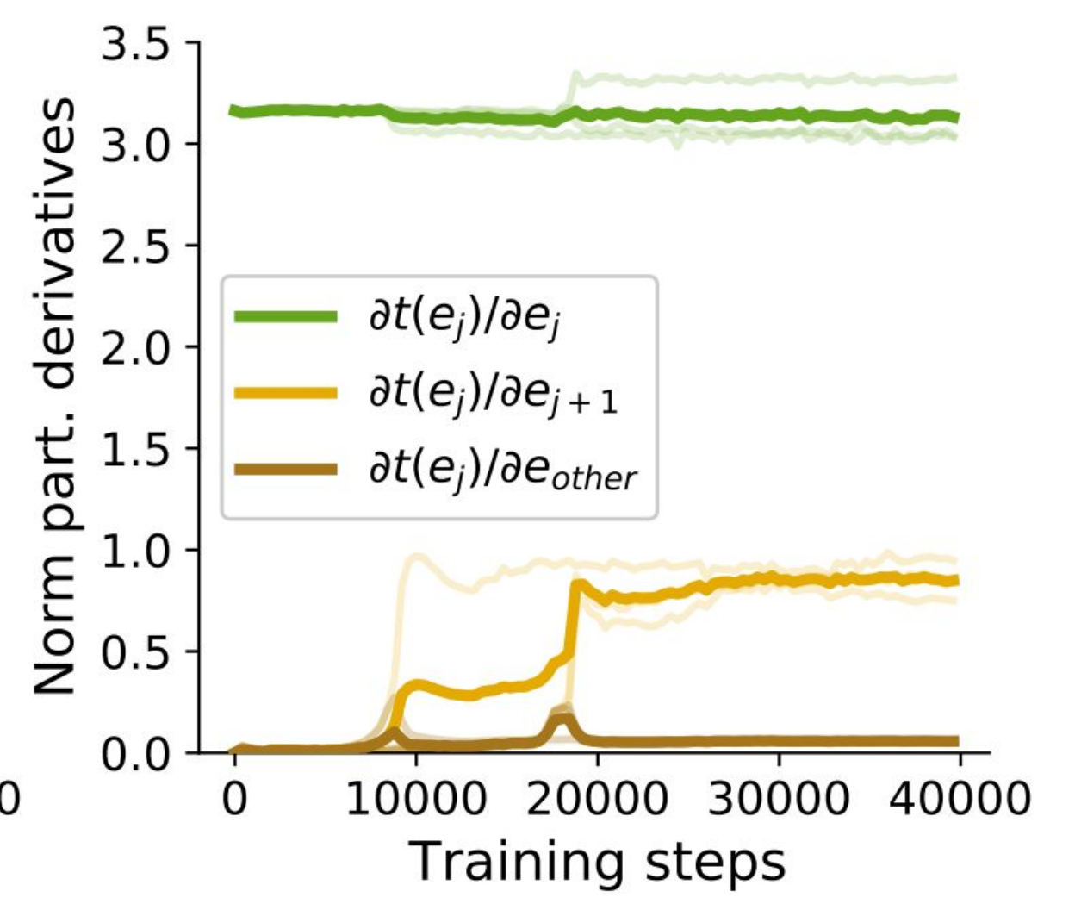


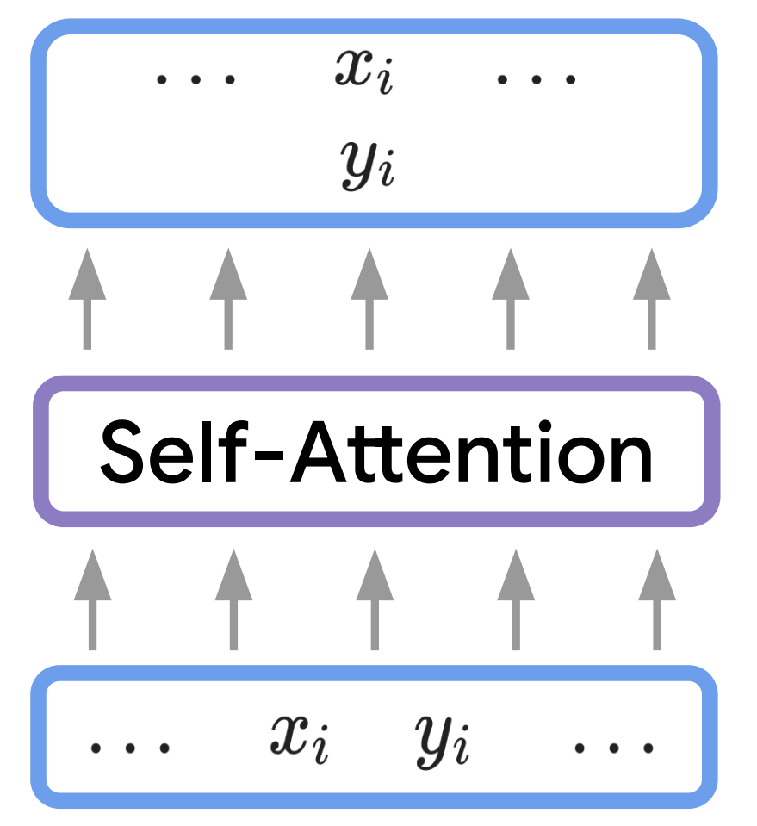


Copying data together

Most of the results in the paper assume tokens consist of concatenated inputs and targets. To relax this assumption, we show that Transformers can learn to construct this on their own to implement GD.







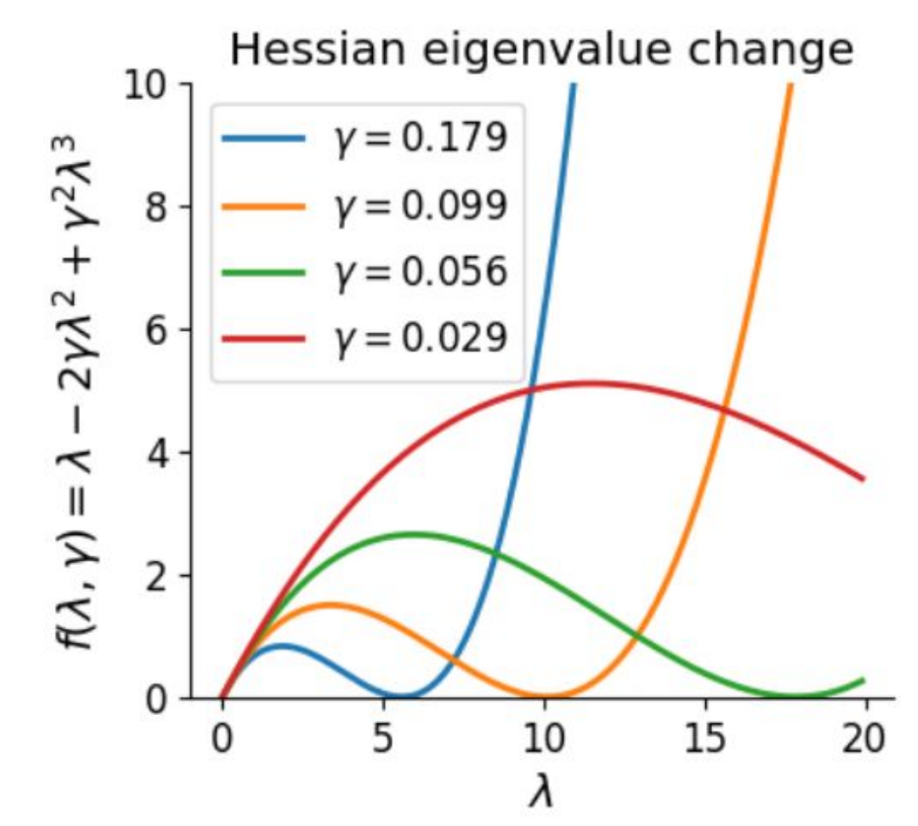
$\ldots \quad x_i \quad \ldots$
y_i
Self-Attention
$\ldots \quad x_i \quad y_i \quad \ldots$

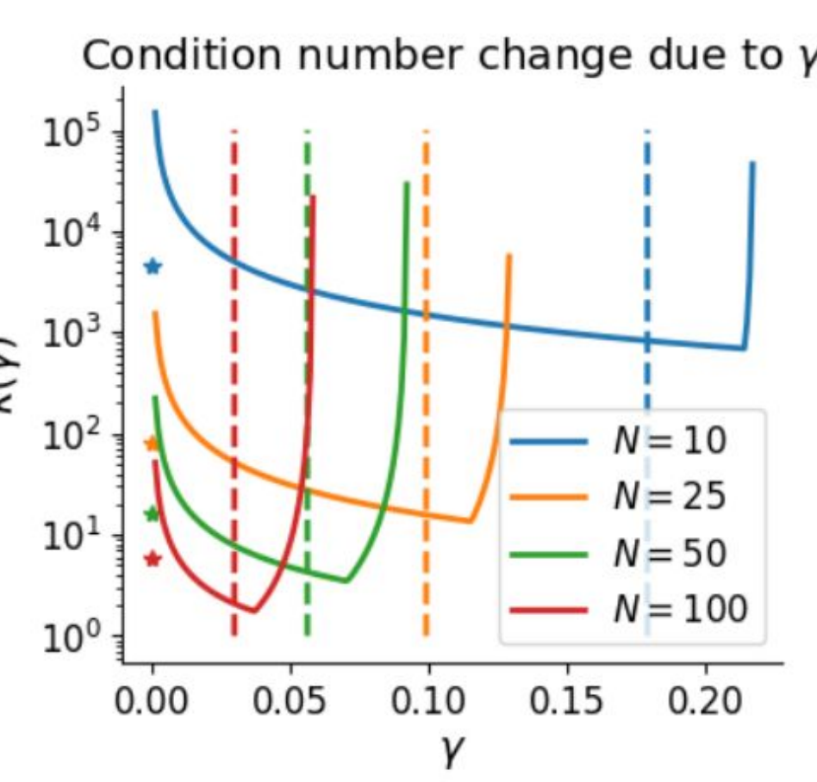
Analyses of GD++

Transformers iteratively transform the input data X while simultaneously doing GD steps. This leads to a change of the loss hessian H and therefore faster learning by better conditioned optimization problems.

$$x_i \leftarrow x_i - \gamma XX^T x_i$$

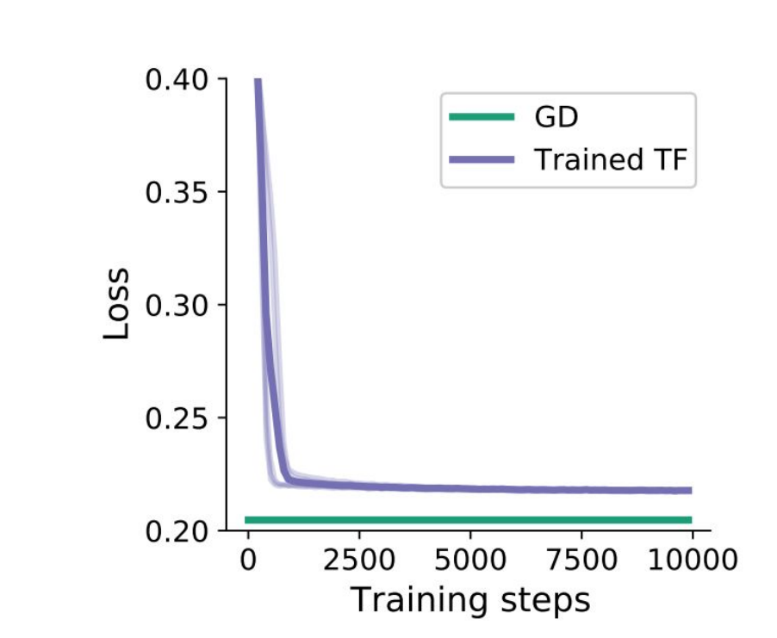
$$H = XX^T = U\Sigma U^T \quad \text{vs} \quad H^{++} = U(\Sigma - 2\gamma\Sigma^2 + \gamma^2\Sigma^3)U^T$$

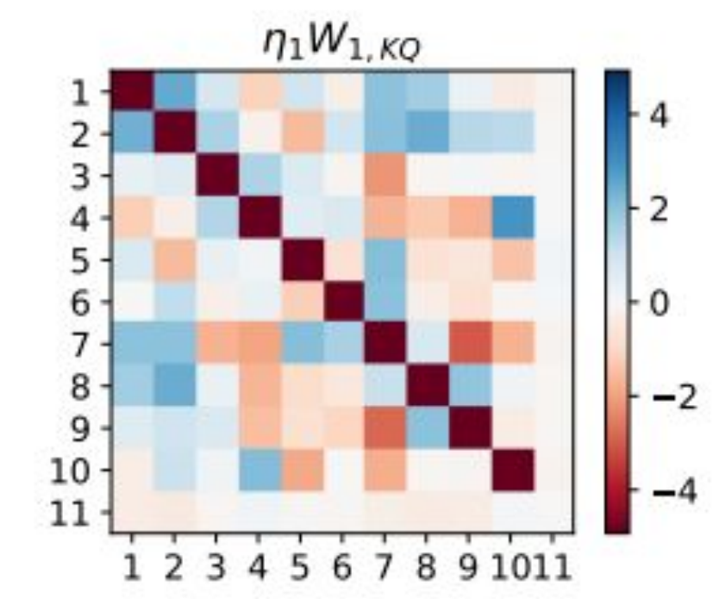


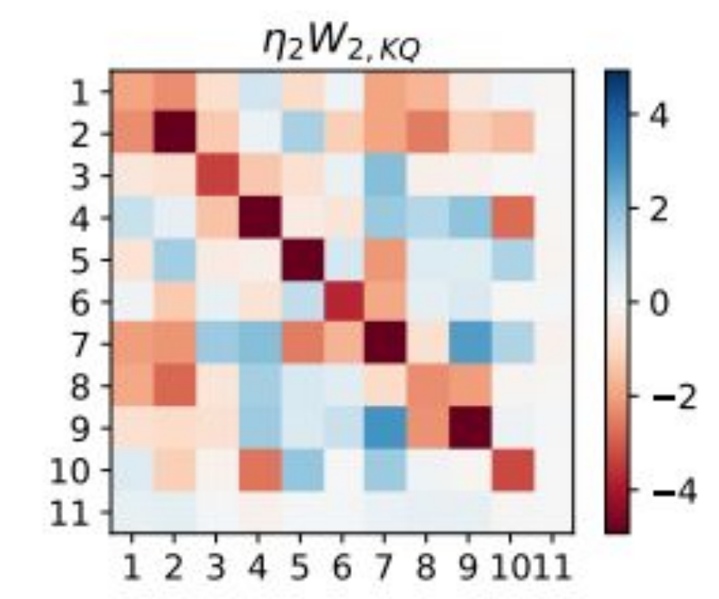


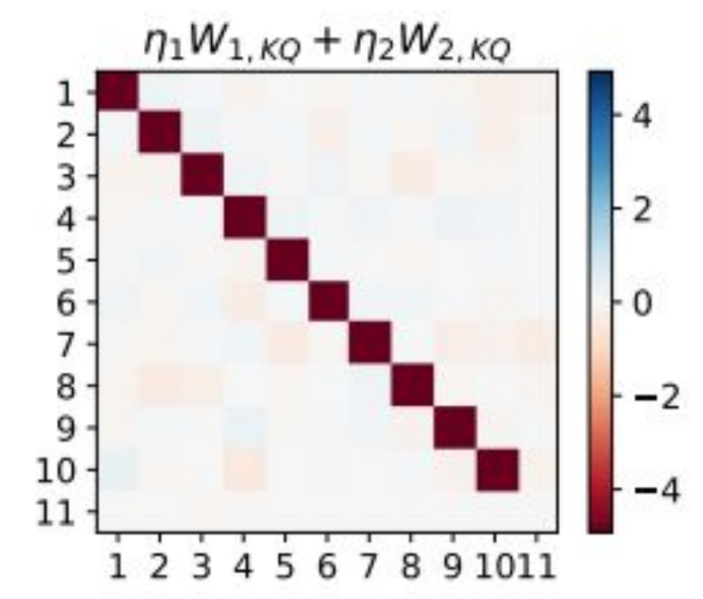
Linearization of softmax self-attention

Multi-head softmax self-attention layers can linearize themselves to approximate a step of GD in a similar fashion.









References

- Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computation 1992.
- Garg, S., Tsipras, D., Liang, P., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), NeurIPS, 2022.
- Akyurek, E., Schuurmans, D., Andreas, J., Ma, T., and ¨ Zhou, D. What learning algorithm is in-context learning? investigations with linear models. ICLR, 2023.