Learning without training: The implicit dynamics of in-context learning

Benoit Dherin[*]
Google Research
dherin@google.com

Michael Munn[*]
Google Research
munn@google.com

Hanna Mazzawi[*]
Google Research
mazzawi@google.com

Michael Wunder
Google Research
mwunder@google.com

Javier Gonzalvo
Google Research
xavigonzalvo@google.com

Abstract

One of the most striking features of Large Language Models (LLM) is their ability to learn in context. Namely at inference time an LLM is able to learn new patterns without any additional weight update when these patterns are presented in the form of examples in the prompt, even if these patterns were not seen during training. The mechanisms through which this can happen are still largely unknown. In this work, we show that the stacking of a self-attention layer with an MLP, allows the transformer block to implicitly modify the weights of the MLP layer according to the context. We argue through theory and experimentation that this simple mechanism may be the reason why LLMs can learn in context and not only during training. Specifically, we show under mild simplifying assumptions how a transformer block implicitly transforms a context into a low-rank weight-update of the MLP layer.

1 Introduction

Large language models and the transformer architecture [1] have revolutionized the field of machine learning, and are about to do the same in many areas of industry, science, and art. In spite of this extensive impact, the mechanisms through which LLM acquire the emergent properties that make them so useful remain for the large part a theoretical mystery [2]. In this work we focus on an LLM ability to learn in context [3, 4], after the training has been fully completed, from examples not seen during training, but provided to the trained system through the prompt. Historically, in machine learning, the ability to extract patterns from a series of examples has been understood as a dynamical process where the model weights are updated as the examples are consumed through an optimization procedure [5]. However, in the case of In-Context-Learning (ICL), there is no immediate explicit weight update that could explain the emergent dynamical nature of trained LLMs that seem to re-organize or reconfigure themselves at the instruction of a user prompt. This mysterious and extremely helpful property of LLMs has led researchers to conjecture an implicit form of weight updates taking place at inference time when a prompt is consumed [6–11]. Recent works have even been able to show that toy models of transformer blocks implicitly performs a sort of gradient descent optimization [7, 9, 10].

In this work, we follow this intuition of implicit weight updates, but we take an opposite approach. Instead of going down the abstraction road and focusing on tractable toy models, we go up that road, and abstract away the contextual property we believe is key in an attention layer. This leads us to a

[*]These authors contributed equally to this work

generalization of a transformer block, which we call a contextual block. We show that layers with this contextual property, when stacked with standard neural networks, implicitly transform a context into a weight update of the very first layer of the stacked neural network. We provide an explicit formula for this implicit update, which turns out to be a rank 1 matrix. This suggests that contextual layers, like self-attention layers, followed by a neural network, perform a sort of implicit fine-tuning of the MLP weights, where the update is computed from the context.

Main contributions:

- We introduce the notion of a contextual block formed by a contextual layer stacked with a neural network generalizing transformer blocks
- We show that for contextual blocks a token output in the presence of a context coincides with the neural network output for that token without context but with its weight matrix updated by a low-rank matrix
- We provide an explicit formula for the neural-network implicit weight-update corresponding to the effect of the context
- We show that token consumption corresponds to an implicit gradient descent learning dynamics on the neural network weights

1.1 Related work

After a certain scale language models are able to learn from examples given in the prompt. This was made clear already from GPT-3 on [3]. This emergent capability is called In-Context-Learning (ICL) [4]. In [12] the authors pose the fundamental question as whether true learning at inference time actually occurs in ICL, or rather the examples in the context help the model retrieve capabilities already learned during pre-training, with no actual new learning taking place at inference time. In fact, [13] posits that the examples in the prompt only serve as a form of Bayesian conditioning rather than true learning. In the same direction, [14] shows that replacing the example labels in the prompt by random labels does not dramatically decrease the performance of ICL, which supports the argument that pre-training capabilities are retrieved through the examples in the prompt rather than learned at inference time. However, revisiting these ideas [15] shows that while this is true for small models, large models start to actually learn from the randomly switched labels in the prompt. Similarly, [16] shows that the emergence of true ICL seems to also be dependent on the data diversity during pre-training in the context of large language models.

On the other side of scale, [8] shows that transformers pre-trained on regression tasks can learn in context new functions as diverse as linear functions, decision trees, two-layer neural nets at inference time. These experiments provide a controlled setting where true ICL can be effectively tested, and they point toward that transformers can learn a form of meta-optimizers as described in [17]. This hypothesis is tested in [6] where (in the same controlled setting of regression tasks) a linear-attention transformer trained with gradient flow is shown to converge toward a meta-optimizer, behaving like gradient descent. Concurrently [7], [9], and [10] exhibit theoretical mechanisms through which example consumption through prompt at run time can be implicitly identified with gradient descent steps on a least-square loss, through implicit weight updates. Recently, [11] showed in the same context that chain-of-thought prompting in that context has the effect of multi-step gradient descent updates. However, all of these implicit learning dynamics hold only under the narrow assumptions of linear layers and prompts constructed of regression example-pairs. These assumptions have been criticized in [18], [19], as not realistic and these authors showed discrepancies between ICL and true gradient descent with a fine-tuning loss constructed from the prompt examples (with [20] recently showing a generalization advantage of ICL over fine-tuning).

In this work, we pursue the idea of ICL being implemented by implicit weight updates corresponding to some form of implicit learning dynamics. However, we remove the limiting assumptions of the linear-attention layer and allow for completely generic prompts, bringing the theory much closer to the actual transformer blocks used in practice. The trade-off is that the implicit dynamics must be understood as taking place on the MLP layer rather than on the self-attention layer of a transformer block, as is the case in [7], [9], and [10].

2 Contextual Blocks

In this section, we abstract some key properties of transformers. In particular, we introduce the notion of contextual layer, which generalizes the self-attention layer of transformer blocks. In this setting a contextual block is the composition of a contextual layer with a standard neural network generalizing the notion of a transformer block. Then we prove our main theorem, which shows that the context for contextual blocks acts as a low-rank fine tuning update of the neural network weights. For the sake of simplicity, we state our results in the case of a neural network without skip-connection. The skip-connection case is similar but more complicated and fully worked out in Appendix A.

We call a *contextual layer*, a network layer $A(\cdot)$ that can take a single vector x alone as input yielding an output $A(x)$; or, optionally, A can take in addition a context C (e.g., a sequence of tokens, an image, etc.) along with the vector x, yielding the output $A([C, x])$. Note, going forward we will often suppress the concatenation notation $[C, x]$ when writing the input to a contextual layer and simply write $A(C, x)$ to denote $A([C, x])$.

As a prototypical and guiding example of a contextual layer, consider the self-attention layer of a transformer block, where the context C is an instruction prompt consisting of a sequence of context tokens $C = [c_1, \dots, c_n]$ and x is the query token from which the LLM will make a prediction. Together C and x create a contextualized input prompt $[C, x] = [c_1, \cdots, c_n, x]$, which is the concatenation of the context tokens and the query token. We take $A(C, x)$ to be the output of the self-attention layer corresponding to last token x. In this way, both $A(C, x)$ and $A(x)$ occupy the same output vector space. Contextual layers produce contextual vectors which we can use to define the difference $\Delta A(C) := A(C, x) - A(x)$ between the layer output with and without context.

Motivated by this generalization of the self-attention layer as a contextual layer, we now generalize the notion of a full transformer block to define the notion of a *contextual block*:

Definition 2.1. *A* contextual block *is the composition $T_W = M_W \circ A$ consisting of a contextual layer A as above with a neural network M_W; i.e, $M_W(z) = f_\theta(Wz + b)$, where W and b are the weights of a the first fully-connected dense layer and $f_\theta(z)$ is the rest of the neural network.*

The following theorem tells us that a contextual block transforms a part $Y \subset C$ of a context C into an implicit update of the neural network weights so that W becomes $W + \Delta W(Y)$ where the information contained in Y has been transferred to the weights through the update $\Delta W(Y)$. In a sense, contextual layers *load* the network weights suitable to the context part Y by implicitly adding a low-rank weight update $\Delta W(Y)$ to the neural network weights W. Namely, the output of the contextual block with the full context C coincides with the contextual block output on the context $C\backslash Y$[1] where Y was removed from C but added to the weight by the update $\Delta W(Y)$.

Theorem 2.2. *Consider a contextual block $T_W = M_W \circ A$ as above formed by a contextual layer A composed with a fully-connected layer M_W with weight matrix W. Given a context C and an input x, the effect of some portion $Y \subset C$ of the context C on the output of the contextual block implicitly corresponds to a rank 1 weight update $W + \Delta W(Y)$ of the first layer of M_W. Namely,*

$$T_W(C, x) = T_{W+\Delta W(Y)}(C\backslash Y, x) \quad \text{where } \Delta W(Y) = \frac{(W\Delta A(Y))A(C\backslash Y, x)^T}{\|A(C\backslash Y, x)\|^2}, \tag{1}$$

where $\Delta A(Y) = A(C, x) - A(C\backslash Y, x)$ is the context vector associated to Y. Note $\Delta W(Y)$ is rank 1, since $W\Delta A(Y)$ is a column vector and $A(C\backslash Y, x)^T$ is a row vector.

Proof. The result follows by direct computation in which we use the notation $M_W(z) = f_\theta(Wz + b)$, where W and b are the weights of the first dense layer of M, while f_θ is the rest of the network. In the notation above, we have by definition that

$$\begin{aligned} T_{W+\Delta W(Y)}(C\backslash Y, x) &= M_{W+\Delta W(Y)}\Big(A(C\backslash Y, x)\Big) && (2)\\ &= f_\theta\Big((W + \Delta W(Y))A(C\backslash Y, x) + b\Big) && (3)\\ &= f_\theta\Big(WA(C\backslash Y, x) + \Delta W(Y)A(C\backslash Y, x) + b\Big). && (4) \end{aligned}$$

[1] $C\backslash Y$ denotes the set of all elements in C which are not in Y.

Replacing now $\Delta W(Y)$ by its definition given in Eq. (1) and using that $\frac{z^T}{\|z\|^2}z = 1$, we obtain

$$\begin{aligned} T_{W+\Delta W(Y)}(C\backslash Y, x) &= f_\theta\Big(WA(C\backslash Y, x) + \frac{(W\Delta A(Y))A(C\backslash Y, x)^T}{\|A(C\backslash Y, x)\|^2}A(C\backslash Y, x) + b\Big) & (5)\\ &= f_\theta\Big(W\big(A(C\backslash Y, x) + \Delta A(Y)\big) + b\Big) & (6)\end{aligned}$$

Now since by definition of the context vector we have that $A(C\backslash Y, x) + \Delta A(Y) = A(C, x)$, we finally get that

$$T_{W+\Delta W(Y)}(C\backslash Y, x) = f_\theta\Big(WA(C, x) + b\Big) = M_W\Big(A(C, x)\Big) = T_W(C, x) \tag{7}$$

which ends the proof. □

Remark 2.3. *Our theorem states that* any *contextual layer produces an implicit weight transfer from the prompt to the first neural network layer, implicitly modifying the behavior of the pre-trained neural network. Among all possible contextual layers (e.g. self-attention, RNN, or recurrent layers with local attention like in [21]), some may be better at providing useful weight modifications than others. It may be interesting to evaluate the generative power of a contextual-layer in terms of the particular form of the implicit weight updates given by our theorem and the special structure of A given by the contextual layer.*

Note that when $Y = C$ is the full context, the theorem above gives a formula to put all the context information into the weight matrix W; namely:

Corollary 2.3.1. *In the notation above, we have that the full context C can be transferred to the neural network weights by the following update:*

$$T_W(C, x) = T_{W+\Delta W(C)}(x), \quad \textit{with} \quad \Delta W(C) = \frac{(W\Delta A)A(x)^T}{\|A(x)\|^2}, \tag{8}$$

where $\Delta A = A(C, x) - A(x)$ is the context vector and ΔW is rank 1, since $W\Delta A$ is a column vector and $A(x)^T$ is a row vector.

Remark 2.4. *The weight transfer formula in Eq.* (1) *can be also rewritten using union/concatenation of context by setting $D = C\backslash Y$; namely:*

$$T_W(D \cup Y, x) = T_{W+\Delta W(Y)}(D, x). \tag{9}$$

In Appendix A, we generalize Theorem 2.2 for neural networks M with skip-connections, as is usually the case for standard transformer blocks. In Section 4, we verify our theoretical results experimentally on a standard concrete example.

3 The implicit learning dynamics of ICL

When the context $C = [c_1, \dots, c_n]$ is a sequence of tokens, an iterative application of Corollary 2.3.1 uncovers an implicit learning dynamics generated by the effect of each context token on the contextual block output. Namely, starting with the initial weight W_0 for the first dense layer of the neural network M_W, we can consider the weight updates corresponding to the incremental addition of a token to the context

$$\begin{aligned} T_{W_0}(c_1, x) &= T_{W_0+\Delta W_0(c_1)}(x)\\ T_{W_0}(c_1, c_2, x) &= T_{W_0+\Delta W_0(c_1,c_2)}(x)\\ &\vdots\\ T_{W_0}(c_1, \dots, c_n, x) &= T_{W_0+\Delta W_0(c_1,\dots,c_n)}(x)\end{aligned}$$

which gives us the following sequence of context weights

$$\begin{aligned} W_1 &= W_0 + \Delta W_0(c_1) & (10)\\ W_2 &= W_0 + \Delta W_0(c_1, c_2) & (11)\\ &\vdots & (12)\\ W_n &= W_0 + \Delta W_0(c_1, \dots, c_n) & (13)\end{aligned}$$

which converges by construction to the effect of the full context on the MLP weight; namely

$$T_{W_n}(x) = T_{W_0}(c_1, \dots, c_n). \tag{14}$$

The following proposition shows that this implicit learning dynamics is a similar of that of online gradient descent, where the tokens play the role of the data points, with a loss changing at each step depending of the token considered at that step:

Proposition 3.1. *In the notation above, the iterative process of weight updates can be realized as a form of stochastic gradient updates*

$$W_i = W_{i-1} - h\nabla_W L_i(W_{i-1}) \tag{15}$$

with learning rate given by $h = 1/\|A(x)\|^2$ and loss at step i given by

$$L_i(W) = \operatorname{trace}(\Delta_i^T W), \tag{16}$$

where $\Delta_i = W_0\Big(A(c_1, \dots, c_i, x) - A(c_1, \dots, c_{i+1}, x)\Big)A(x)^T$.

Proof. First of all, considering the sequence of W_i's defined in Eq. (10)-(13) above, we have that

$$\begin{aligned} W_{i+1} - W_i &= \Delta W_0(c_1, \dots, c_{i+1}) - \Delta W_0(c_1, \dots, c_i) && (17)\\ &= \frac{W_0\Big(A(c_1, \dots, c_{i+1}, x) - A(c_1, \dots, c_i, x)\Big)A(x)^T}{\|A(x)\|^2} && (18)\\ &= -h\Delta_i, && (19)\end{aligned}$$

with $h = 1/\|A(x)\|^2$ and $\Delta_i = W_0\Big(A(c_1, \dots, c_i, x) - A(c_1, \dots, c_{i+1}, x)\Big)A(x)^T$.

This means that

$$W_{i+1} = W_i - h\Delta_i = W_i - h\nabla_W \operatorname{trace}(\Delta_i^T W), \tag{20}$$

since in general we have $\nabla_W \operatorname{trace}(A^T W) = A$. □

Notice that Δ_i measures the effect of the addition of context token c_{i+1} to the partial context $c_1, \dots, c_i$. When c_i has no effect on the output, that is, when $A(c_1, \dots, c_i, x) - A(c_1, \dots, c_{i+1}, x)$ is zero, and the corresponding update $\nabla_W L_i(W) = \Delta_i$ vanishes. Figure 2 shows in a simple experiment that these gradients vanish as the learning dynamics converges toward the whole context.

Remark 3.2. *Interestingly, it is possible to derive a different but similar implicit learning dynamics $W_0, W_1, \dots, W_n$ by considering partial updates leaving the overall contextual block output unchanged at each step when the partial updates are used in conjunction with the remaining tokens: $T_{W_i}(c_{i+1}, \cdots, c_n, x) = T_{W_0}(c_1, \dots, c_n, x)$. This dynamics is described in Appendix B. The difference is that it can in general no longer be represented by a gradient updates, but leads to a factorization formula for the overall weight W_n such that $T_{W_n}(x) = T_{W_0}(c_1, \dots, c_n, x)$.*

4 Experiments

In order to verify Theorem 2.2 in practice, we consider a well-defined problem of learning a function class from *in-context* examples. This task has been studied independently in [6, 22]. In those works, the authors show that it is possible to train a transformer from scratch to perform in-context learning of linear functions. That is to say, given a transformer model trained on a class of linear functions, the trained model is able to learn new and unseen linear functions (drawn from a distribution similar to that used during training) purely from in-context examples with performance comparable to the optimal least squares estimator.

In [6, 22], the authors were concerned with quantifying how robust transformers are (and are not) to distributional shifts between the training data of the model and inference-time prompts. That is not our goal here. Instead, since these works have have already verified that transformers can indeed learn linear models in-context, we use here a similar experimental framework to verify that the in-context prompts can effectively be transferred to a weight update via Equation (8). We verify that the prediction made by the trained model with an in-context prompt is identical to the prediction made by the model with MLP weights modified according to Equation (8) but without access to the in-context prompt.

4.1 Setup

At a high-level, similar to [6], we train a simple transformer on instances of prompts of input-output pairs of the form $(x_1, h(x_1), \ldots, x_N, h(x_N), x_{\text{query}})$ where the x_i, x_{query} are sampled i.i.d. from a distribution $\mathcal{D}_x$ and the function h is sampled independently from a distribution over functions in a function class $\mathcal{H}$. In particular, we take $\mathcal{H}$ to be the class of linear functions so that $h(x) = \langle w, x \rangle$ and where $x_i, x_{\text{query}}, w \sim \mathcal{N}(0, I_d)$. The goal of the in-context learner is to use the input-output pair prompt as above and to form a prediction $\widehat{y}(x_{\text{query}})$ so that $\widehat{y}(x_{\text{query}}) \approx h(x_{\text{query}})$.

Each training prompt is indexed by a task denoted $\tau \in \mathbf{N}$ and takes the form $P_\tau = (x_{\tau,1}, h_\tau(x_{\tau,1}), \ldots, x_{\tau,N}, h_\tau(x_{\tau,N}), x_{\tau,\text{ query}})$. We can express each prompt as an embedding matrix E_τ so that

$$E_\tau := \begin{pmatrix} x_{\tau,1} & x_{\tau,2} & \cdots & x_{\tau,N} & x_{\tau,\text{ query}} \\ \langle w_\tau, x_{\tau,1} \rangle & \langle w_\tau, x_{\tau,2} \rangle & \cdots & \langle w_\tau, x_{\tau,N} \rangle & 0 \end{pmatrix} \in \mathbb{R}^{(d+1)\times(N+1)}.$$

Using the notation in Section 2, we can think of E_τ as a contextualized input prompt where

$$C = [c_1, \ldots, c_N] = \begin{pmatrix} x_{\tau,1} & x_{\tau,2} & \cdots & x_{\tau,N} \\ \langle w_\tau, x_{\tau,1} \rangle & \langle w_\tau, x_{\tau,2} \rangle & \cdots & \langle w_\tau, x_{\tau,N} \rangle \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} x_{\tau,\text{query}} \\ 0 \end{pmatrix}$$

so that $E_\tau = (C, x)$. Let θ denote the parameters of the model. The model prediction $\widehat{y}(x_{\tau,\text{ query}})$ for the token $x_{\tau,\text{query}}$ is the last component of the query-token output by a single transformer block[2], that is,

$$\widehat{y}(x_{\tau,\text{ query}}) = T_W(C, x)_{(d+1)} \tag{21}$$

Note that, defined this way, the dimensionality of $T_W(C, x)$ and $T_{W+\Delta W}(x)$ agree. We train the transformer using the loss over a batch of size B defined as

$$\widehat{L}(\theta) = \frac{1}{2B} \sum_{\tau=1}^{B} \left(\widehat{y}_{\tau,\text{query}} - \langle w_\tau, x_{\tau,\text{query}} \rangle \right)^2.$$

4.2 Verifying Theorem 2.2

Given a transformer trained on linear functions, we show that the in-context prompt can be transferred to a weight update as defined in Equation (8). Namely we want to show that

$$T_W(C, x) = T_{W+\Delta W}(x);$$

or equivalently,

$$T_W\left(\begin{pmatrix} x_{\tau,1} & x_{\tau,2} & \cdots & x_{\tau,N} & x_{\tau,\text{query}} \\ \langle w_\tau, x_{\tau,1} \rangle & \langle w_\tau, x_{\tau,2} \rangle & \cdots & \langle w_\tau, x_{\tau,N} \rangle & 0 \end{pmatrix}\right) = T_{W+\Delta W}\left(\begin{pmatrix} x_{\tau,\text{query}} \\ 0 \end{pmatrix}\right)$$

where ΔW is computed as in Equation (8). Figure 1 compares the validation loss obtained by using each side of the equation above to make predictions upon an evaluation query-token. The loss values for both setups are reported for each checkpoint obtained during pre-training. We can see that these losses are in remarkable agreement with each other, even on a zoomed-in version of the graph (right).

4.3 Convergence of ΔW

We conduct some experiments to understand how the weights adapt as the in-context prompt is processed by the model during the implicit learning dynamics described by Proposition 3.1. In particular, we want to verify that the gradient updates vanish as context convergence is reached.

We create a sequence $\{(\Delta W)_i\}_{i=1}^N$ where each $(\Delta W)_i$ is as described in Equations (10)-(13). That is we have that

$$T_W(C_i, x) = T_{W+(\Delta W)_i}(x)$$

where

$$C_i = [c_1, \ldots, c_i] = \begin{pmatrix} x_{\tau,1} & \cdots & x_{\tau,i} \\ \langle w_\tau, x_{\tau,1} \rangle & \cdots & \langle w_\tau, x_{\tau,i} \rangle \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} x_{\tau,\text{query}} \\ 0 \end{pmatrix}.$$

[2]For the sake of simplicity, in Theorem 2.2 and in this experiment, we use standard transformer blocks [1] but without the skip connection on the MLP layer; see Appendix A to learn how to deal with the skip connection.

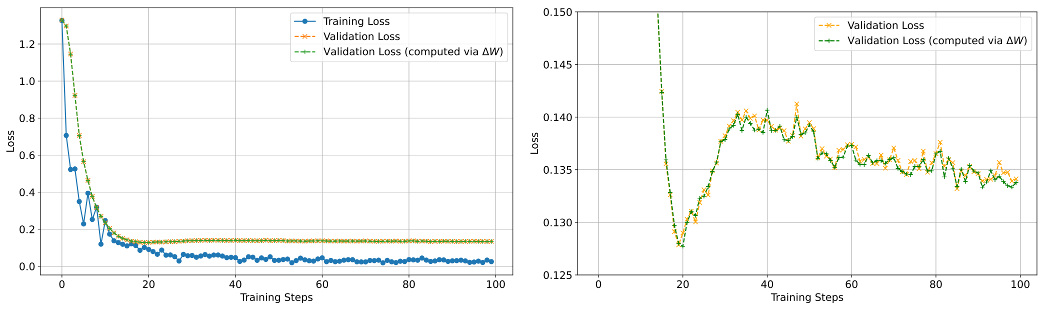

Figure 1: Train and Validation loss curves. Here, the "Validation loss (computed via ΔW)" refers the loss computed using $T_{W+\Delta W}(x)$; i.e., the trained model prediction given only x_{query} but with MLP weights modified by ΔW as defined in Equation (8). **Left:** Training loss and both validation Loss curves. **Right:** Close-up of validation loss computed both ways; i.e., using $T_W(C, x)$ vs. $T_{W+\Delta W}(x)$.

If we let W_0 denote the learned weights of the first dense layer, it follows from Corollary 2.3.1, that for any $i = 1, 2, \ldots, N$

$$(\Delta W)_i = \frac{(W_0 \Delta A_i) A(x)^T}{\|A(x)\|^2}, \quad \text{where } \Delta A_i := A(c_1, \ldots, c_i, x) - A(x).$$

Intuitively, we expect that as the 'in-context learner' processes more of the prompt, the relative change in the $(\Delta W)_i$ should decrease. In Figure 2 we verify that this is indeed the case.

For a given context $C_i = [c_1, \ldots, c_i]$ of length i, we plot the marginal change in $(\Delta W)_i$ from incorporating one additional context token c_{i+1} which would yield $(\Delta W)_{i+1}$ for the context $C_{i+1} = [c_1, \ldots, c_i, c_{i+1}]$. We measue this marginal change via L2-norm; i.e., for each context length i we plot on the y-axis the quantity corresponding to the gradient updates of Proposition 3.1, namely:

$$\|\nabla_W L_i(W)\|_2 = \|(\Delta W)_{i+1} - (\Delta W)_i\|_2.$$

We observe in Figure 2 that the gradient updates decrease and vanish as the implicit learning dynamics progresses toward the full context as we expect from a converging gradient descent dynamics.

4.4 Comparison with Finetuning

We pretrain a transformer model (with a standard single transformer block without MLP skip-connection) with examples of the form

$$E_\tau := \begin{pmatrix} x_{\tau,1} & x_{\tau,2} & \cdots & x_{\tau,N} & x_{\tau,\text{query}} \\ \langle w_\tau, x_{\tau,1}\rangle & \langle w_\tau, x_{\tau,2}\rangle & \cdots & \langle w_\tau, x_{\tau,N}\rangle & 0 \end{pmatrix} \in \mathbb{R}^{(d+1)\times(N+1)}.$$

Here we take $d = 2$ and $N = 50$.

For fine-tuning we create one new test example using ω_{test} that the model has never seen during pretraining, though ω_{test} is drawn from the same distribution that each ω_τ is drawn from during pretraining. Call this example $\mathcal{D}_{FT}$:

$$\mathcal{D}_{FT} = \begin{pmatrix} x_1 & \cdots & x_M & x_{\text{test}} \\ \langle \omega_{\text{test}}, x_1\rangle & \cdots & \langle \omega_{\text{test}}, x_M\rangle & 0 \end{pmatrix}$$

Now for each $i = 1, 2, \cdots, M$ we create a dataset for finetuning by taking the first i element of $\mathcal{D}_{FT}$, ignoring the last column which is our test query. That is, for all $i = 1, \cdots, M$

$$\mathcal{D}^i_{FT} = \begin{pmatrix} x_1 & x_2 & \cdots & x_i \\ \langle \omega_{\text{test}}, x_1\rangle & \langle \omega_{\text{test}}, x_2\rangle & \cdots & \langle \omega_{\text{test}}, x_i\rangle \end{pmatrix}$$

We initialize the transformer with the pretrained weights, then finetune using stochastic gradient descent with learning rate 0.01 taking one example at a time in the same order that they are processed in-context. During finetuning we only update the weight matrix of the MLP layer. So, for each

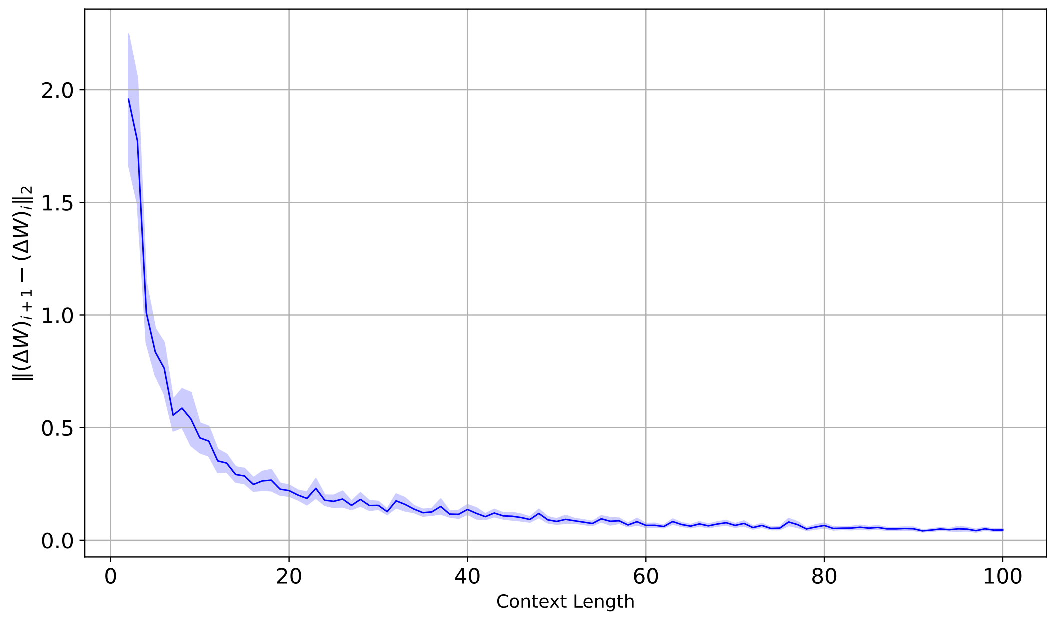

Figure 2: Convergence of $(\Delta W)_i$. As more of the context in processed, the relative change in the weights W converges to zero. For context length $i > 2$, the plot above represents the average difference $\|(\Delta W)_{i+1} - (\Delta W)_i\|_2$ and the standard error over 100 separate trials.

$i = 1, \ldots, M$, we perform i steps of gradient descent with batch size of 1. After finetuning on all M examples we compute the loss of the finetuned model on the test query $(x_{\text{test}}, 0)$. We call this the 'gradient descent (GD) test loss' for i steps.

Similarly, for each i we compute the weight transfer as defined in Equation 8 using the context

$$C_i = \begin{pmatrix} x_1 & x_2 & \cdots & x_i \\ \langle \omega_{\text{test}}, x_1 \rangle & \langle \omega_{\text{test}}, x_2 \rangle & \cdots & \langle \omega_{\text{test}}, x_i \rangle \end{pmatrix}$$

and the same test query as before $x = (x_{\text{test}}, 0)$. Using the value of ΔW from the weight transfer formula, we compute the loss on $(x_{test}, 0)$. We call this the 'ΔW test loss' for context length i.

In the Figure 3 below we plot the finetune gradient descent test loss against the ΔW weight transfer test loss. The plot displays the average over 100 separate trials. Although different, we see that the two learning processes (fine-tuning and implicit weight-update dynamics) minimize the loss in similar ways.

5 Conclusion and Limitations

Our approach to the transformer block mechanics underpinning ICL improves over previous approaches in the sense that it does not put any restriction on the self-attention layer architecture in order to extract a form of implicit learning dynamics in weight space. Previous theoretical works focusing on the internals of transformers were able to derive such an implicit learning dynamics but only under very limiting assumptions on the self-attention layer (like linear attention, or/and a single-head; see [9–11, 19]). In fact our results remain valid if the self-attention layer is switched by other forms of contextual layers, like that of a RNN, or any layer that can take an input and optionally a context. This is surprising because our analysis hints that ICL is less about the internals of self-attention, but rather about the fact that regular neural networks can transfer modification of input space to their weight structure. This is a deep property that has been noticed in a number of theoretical works, and helped understand why deep neural networks generalize so well [23–25].

However, although closer to reality since we remove any limiting assumptions on the self-attention layer, we are still analyzing a toy model in the following sense, which constitutes the limitation of our analysis:

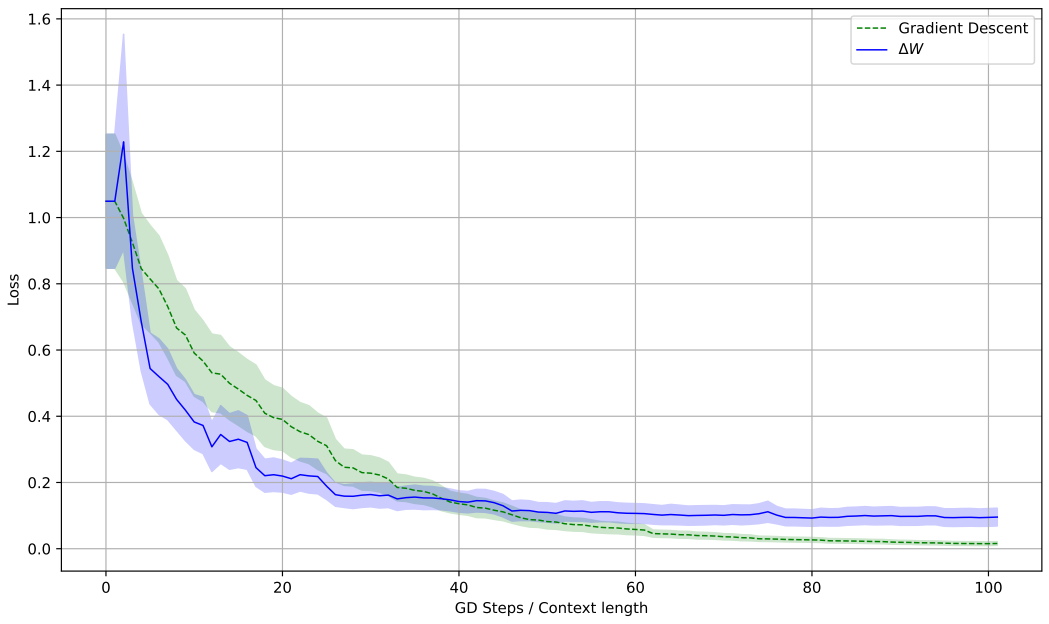

Figure 3:

- Our derivation is valid only for a single transformer block because our main theorem quantifies the effect of the context only on the output of the very last input token, and not on the whole transformer block output.
- Our main theorem analyses the effect of context w.r.t. the first generated token only. It does not capture the full mechanics of generation beyond that.

In spite of these limitations, we hope this work will help understand better the mysterious phenomena that emerge at inference time for LLMs.

6 Acknowledgments

We would like to thank Corinna Cortes, Sanjiv Kumar, Michael Riley, Sergei Vassilvitskii, Sammy Jerome, Peter Bartlett, Spencer Frei, Dilan Gorur, Aranyak Mehta and Daniel Ramage for helpful discussions, suggestions, and feedback during the development of this work.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, pages 5998–6008, 2017.

[2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. urlhttp://www.deeplearningbook.org.

[6] Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024.

[7] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *ICLR*, 2023.

[8] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In *NeurIPS*, 2022.

[9] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 35151–35174, 2023.

[10] Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019. Association for Computational Linguistics, 2023.

[11] Jianhao Huang, Zixuan Wang, and Jason D. Lee. Transformers learn to implement multi-step gradient descent with chain of thought. In *The Thirteenth International Conference on Learning Representations*, 2025.

[12] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021.

[13] 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*, 2022.

[14] 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? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

[15] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently, 2024.

[16] Allan Raventos, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[17] Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems. *ArXiv*, abs/1906.01820, 2019.

[18] Lingfeng Shen, Aayush Mishra, and Daniel Khashabi. Position: Do pretrained transformers learn in-context by gradient descent? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 44712–44740, 2024.

[19] Gilad Deutch, Nadav Magar, Tomer Natan, and Guy Dar. In-context learning and gradient descent revisited. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024.

[20] Yi Lu, Jing Nathan Yan, Songlin Yang, Justin T Chiu, Siyu Ren, Fei Yuan, Wenting Zhao, Zhiyong Wu, and Alexander M Rush. A controlled study on long context extension and generalization in LLMs, 2025.

[21] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[22] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

[23] Benoit Dherin, Michael Munn, Mihaela Rosca, and David Barrett. Why neural networks find simple solutions: The many regularizers of geometric complexity. In *NeurIPS*, 2022.

[24] Cong Ma and Lexing Ying. On Linear Stability of SGD and Input Smoothness of Neural Networks, 2021.

[25] Sihyeon Seong, Yegang Lee, Youngwook Kee, Dongyoon Han, and Junmo Kim. Towards flatter loss surface via nonmonotonic learning rate scheduling. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2018.

[26] Bobby He and Thomas Hofmann. Simplifying transformer blocks. In *ICLR*, 2024.

[27] Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *ICLR*, 2024.

[28] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025.

[29] Guan Zhe Hong, Bhavya Vasudeva, Vatsal Sharan, Cyrus Rashtchian, Prabhakar Raghavan, and Rina Panigrahy. Latent concept disentanglement in transformer-based language models. *arXiv preprint arXiv:2506.16975*, 2025.

A Contextual blocks with skip-connections

We now consider the case of contextual blocks with skip connections encompassing the standard Pre-LN transformer block as for instance described in [26].

Definition A.1. *A contextual block with skip connection is a layer of the form*

$$T(C,x) = x + A(C,x) + W' g_\theta(WA(C,x) + b) + b' \tag{22}$$

where g_θ is any differential model and $A(C,x)$ is a contextual layer.

We can generalize Theorem 2.2 to this context by allowing to update not only the weight matrix W of the first layer but also the bias term b' of the last layer.

Theorem A.2. *Consider a contextual block T with skip connection as above: i.e.,*

$$T(C,x) = x + A(C,x) + W' g_\theta(WA(C,x) + b) + b' \tag{23}$$

where $A(C,x)$ is a contextual layer and $g_\theta(z)$ any differentiable model. Then the effect of part $Y \subset C$ the context C on the output of the contextual block implicitly corresponds to a rank 1 weight update of the first-layer weight matrix $W(Y) = W + \Delta W(Y)$ as well as an update of last-layer bias term $b'(Y) = b' + \Delta b'(Y)$ so that:

$$T_{W,\,b'}(C,x) = T_{W(Y),\,b'(Y)}(C \backslash Y, x), \tag{24}$$

The weight updates are given by the following formula

$$\begin{aligned} \Delta b'(Y) &= \Delta A(Y), & (25)\\ \Delta W(Y) &= \frac{(W\Delta A(Y))A(C\backslash Y,x)^T}{\|A(C\backslash Y,x)\|^2}, & (26) \end{aligned}$$

where $\Delta A(Y) = A(C,x) - A(C\backslash Y,x)$ is the context vector associated to Y and $\Delta W(Y)$ is rank 1, since $W\Delta A(Y)$ is a column vector and $A(C\backslash Y,x)^T$ is a row vector.

Proof. The result follows by direct computation. In the notation above, we have by definition that

$$\begin{aligned} T_{W(Y),\,b'(Y)}(C\backslash Y,x) &= x + A(C\backslash Y,x) + W' g_\theta\Big(\big(W + \Delta W(Y)\big)A(C\backslash Y,x) + b)\Big) + b' + \Delta b'(Y)\\ &= x + A(C\backslash Y,x) + \Delta b'(Y)\\ &\quad + W' g_\theta\Big(WA(C\backslash Y,x) + \Delta W(Y)A(C\backslash Y,x) + b)\Big) + b' \end{aligned}$$

Replacing now $\Delta W(Y)$ by its definition and using that $\frac{z^T}{\|z\|^2} z = 1$, we have that

$$\Delta W(Y)A(C\backslash Y,x) = \frac{(W\Delta A(Y))A(C\backslash Y,x)^T}{\|A(C\backslash Y,x)\|^2} A(C\backslash Y,x) = W\Delta A(Y).$$

Therefore, we get that

$$T_{W(Y),\,b'(Y)}(C\backslash Y,x) = x + A(C\backslash Y,x) + \Delta A(Y) + W' g_\theta\Big(W\big(A(C\backslash Y,x) + \Delta A(Y)) + b)\Big) + b'$$

Now since by definition of the context vector we have that $A(C\backslash Y,x) + \Delta A(Y) = A(C,x)$, we finally get that

$$T_{W(Y),\,b'(Y)}(C\backslash Y,x) = x + A(C,x) + W' g_\theta\Big(WA(C,x) + b)\Big) + b' = T_{W,b'}(C,x)$$

which ends the proof. □

Observe that the bias vector update $\Delta b'(Y)$ bears some similarity in spirit with the function vectors of [27], the transcoder outputs of [28], or the latent concept representations of [29] used to edit a transformer weights. Note also that this theorem is not only valid for contextual layers like Pre-LN transformer blocks as in [26] but also other type of contextual layers as for instance these in the Griffin recurrent models with local attention [21].

B An alternative implicit learning dynamics of ICL

In this section, we describe an alternate implicit learning dynamics following an iterative application of Theorem 2.2. It uncovers an implicit weight update dynamics generated by the effect of each context token on the contextual block output. This means that, while no explicit weight update is performed while a transformer block generates the first response token, the actual output is equivalent to that of the contextual block without context but for which an implicit learning dynamics in weight space has happened. We now describe this learning dynamic. Namely, starting with the initial weight W_0 for the first dense layer of the neural network M_{W_0}:

$$T_{W_0}(c_1, \ldots, c_n, x) = T_{W_0 + \Delta W_0(c_1)}(c_2, \ldots, c_n, x) \tag{27}$$

which gives us the first weight update corresponding on the effect of token c_1 on the first-layer weight matrix:

$$W_1 = W_0 + \frac{(W_0 \Delta A(c_1))A(c_2, \ldots, c_n, x)^T}{\|A(c_2, \ldots, c_n, x)\|^2} \tag{28}$$

If we continue this process iteratively, we obtain the next weight update corresponding to the consumption of the second token:

$$T_{W_1}(c_2, \ldots, c_n, x) = T_{W_1 + \Delta W_1(c_2)}(c_3, \ldots, c_n, x) \tag{29}$$

which yields

$$W_2 = W_1 + \frac{(W_1 \Delta A(c_2))A(c_3, \ldots, c_n, x)^T}{\|A(c_3, \ldots, c_n, x)\|^2} \tag{30}$$

We can summarize this iterative process of implicit weight updates for each successive token:

Corollary B.0.1. *In the notation above, the iterative process of weight updates*

$$W_i = W_{i-1} + \frac{(W_{i-1} \Delta A(c_i))A(c_{i+1}, \ldots, c_n, x)^T}{\|A(c_{i+1}, \ldots, c_n, x)\|^2} \tag{31}$$

starting with the initial weights of the first dense layer W_0 models the transfer of information from the prompt token c_i into the contextual block weights: Namely, we have that

$$T_{W_i}(c_{i+1}, \ldots, c_n, x) = T_{W_0}(c_1, \ldots, c_n, x), \tag{32}$$

for all $i = 1, \ldots, n$ with $\Delta A(c_i) = A(c_i, \ldots, c_n, x) - A(c_{i+1}, \ldots, c_n, x)$.

Notice that $\Delta A(c_i)$ measures the effect of context token c_i on the contextual block output. When c_i has no effect on the output, that is when $\Delta A(c_i)$ is zero, and the corresponding update vanishes. Notice that the weight update at step i is linear in the weights; namely, we can rewrite it as

$$W_i = W_{i-1} + h_i W_{i-1} A_i = W_{i-1}(1 + h_i A_i) \quad \text{where} \quad A_i := \Delta A(c_i) A(c_{i+1}, \ldots, c_n, x)^T \tag{33}$$

with adaptive learning rate given by

$$h_i := \frac{1}{\|A(c_{i+1}, \ldots, c_n, x)\|^2}. \tag{34}$$

In particular, this gives us a factorization formula for the total implicit weight matrix corresponding to the effect of context $[c_1, \ldots, c_n]$ on input-token x:

$$W_n \quad = \quad W_0(1 + h_1 A_1)(1 + h_2 A_2) \cdots (1 + h_n A_n). \tag{35}$$