TDL - Project Report

Gabrielle Caillaud gabrielle.caillaud@student-cs.fr
Damien Bouet damien.bouet@student-cs.fr

January 31, 2025

1 Introduction

In this project, we review the paper [1] that focuses on theoretical aspects of the training of transformers. Transformers have first been introduced by [2] in 2017 and have since then become the state-of-the art architecture in natural language processing. As the use of transformers increase, there is a need to precisely understand how and what these models learn during training, and the paper [1] addresses this issue.

Our Contributions: We describe the work done in the article about associative memory and present the demonstrations. Our personal additional work is mainly about the experiments, in Sections 4 and 5.
All the code for our experiments can be found on our GitHub repo : `https://github.com/gabriellecaillaud/TDL_BirthTransformer`.

2 Overview of the paper

2.1 The simplified transformer setup

The article focuses on decoder-only and autoregressive transformer architectures, which is the traditionnal architecture for GPT-like models [3] (Figure 1).

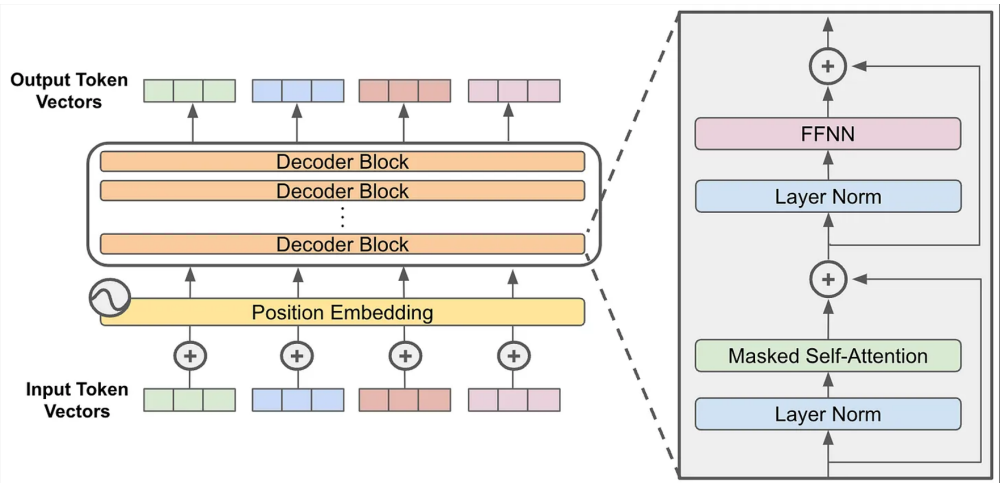


Figure 1: Decoder-only architecture [source: Deep (Learning) Focus]

The article differentiates two types of knowledge: global knowledge and context-specific information. Global knowledge is everything related to syntactic rules, grammar rules or general facts, that do not appear in the prompts and therefore must be stored in the model. Context-specific knowledge is how the transformer predicts the next tokens based on the previous tokens of the prompt.

To precisely analyze the differences between these two types of knowledge, they use a synthetic dataset generated from bigrams models. Data is sampled from a general distribution that emulates general knowledge. However, in each sequence, the transitions are changed for a few trigger tokens so that these trigger tokens are always followed by some output tokens. Because this induces sequence-specific patterns, this models in-context knowledge.

The authors' claim is that transformer models learn by creating associative memories within their weight matrices. To analyze the transformer's architecture, they create a simplified transformer model:

1. They freeze some layers, like the embedding layer, at their random initialization
2. They remove the normalization layers
3. They use shallow transformers, with one or two blocks.
4. They do not use Multi-Head attention, instead they use one head per attention layer
5. For some experiments, they remove the feed-forward layer.

Thus, the complex and usual transformer architecture of Figure 1, becomes the simplest one in Figure 2. In some experiments, we will sometimes use feed-forward layers, giving a slightly more complex architecture. But on the opposite, we will also sometimes simplify it even more, by freezing some of the self-attention layer weights.

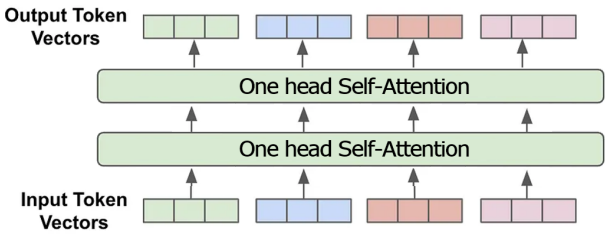


Figure 2: Simplified decoder-only architecture

2.2 What a two layers simple transformer can learn

The authors find that transformer models learn global knowledge relatively quickly during training, primarily through the feed-forward layers. The learning of in-context information is slower, and relies on a mechanism called *induction head*: a combination of two attention heads located in separate layers, that allow the model to predict the next token based on its occurrences in the input sequence. The first head attends to the previous token, copies its embedding to the next token, and the second head attends the embedding copied by the first head to produce the output token.

They claim that the induction head is created via a top-down learning process: First, the output matrix learns associations between trigger tokens and their corresponding outputs. Then, the

attention matrices learn to focus on the relevant key-value pairs to build the memory circuit.

They also claim that the transfomer is able to filter the signal from noise. This noise can come from the addition of the embedding and the positional encoding or residual connection. If the information mixed with the initial signal is not interesting (the position or residual may sometimes not be important/useful), the model must manage to filter and retrieve the signal's information. (See section 3)

3 Associative memory

3.1 What is it?

The core contribution of the paper is on what they call *associative memory*. The author's claim is that when the embeddings are nearly orthogonal, the weight matrices in the attention layers store the memory.

In an attention block, we have four parts: A key matrix W_K, a query matrix W_Q, a value matrix W_V and an output matrix W_O. These trainable matrices are implemented in PyTorch as linear layers without bias. Given an input sequence $x_{1:T} \in \mathcal{R}^{dxT}$ of embeddings, the attention block computes for each t the following vector:

$$x'_t = W_O W_V x_{1:t} \sigma(x_{1:t}^T W_K^T W_Q x_t)$$

where σ is a softmax function. This expression lead us to introduce $W = W_K^T W_Q$,which is the matrix in which memory is stored. In their article, the authors use one head per attention layer for simplification, although transformers usually implement multi-head attention. They also set $W_Q = Id$ in all of their experiments, so that $W = W_K^T W_Q = W_K^T$, and this make it easier in the experiments to study the how the relationship between keys and queries is learned.

If we assume that the key vector u and the query vector v are orthonormal, the matrix W is able to store association between the two via :

$$W = \sum_{i,j} \alpha_{i,j} v_j u_i^T$$

We have indeed $v_j^T W u_i = \sum_{i,j} \alpha_{i,j} v_j^T v_j u_i^T u_i = \alpha_{i,j}$, and $\alpha_{i,j}$ is learned to assess the relevance of the (i, j) pair.

This information stored in the matrix W via this interpretation is called associative memory. Although this requires the vectors u_i and v_j to be nearly-orthonormal, the authors claim that this can be achieved in high dimension d when u_i and v_j are sampled from a Gaussian distribution of variance $\frac{1}{d}$. To ensure this in the experiments, the embeddings layers are initialized randomly as a Gaussian distribution of variance $\frac{1}{d}$ and are frozen after initialization.

3.2 Some theoretical results

Lemma 1: Let p be a data distribution over input-output tokens, and consider the following loss, where the input and output embeddings W_E and W_U are fixed:

$$L(W) = \mathbb{E}_{(z,y)\sim p}[\ell(y, W_U W w_E(z))]$$

with ℓ the cross-entropy loss. The gradients of the population loss L then take the form:

$$\nabla_W L(W) = \sum_{k=1}^{N} \mathbb{E}_z[(\hat{p}_W(y=k|z) - p(y=k|z))w_U(k)w_E(z)^T]$$

where $\hat{p}_W(y=k|x) = \sigma(W_U W w_E(z))_k$ are the model's predicted probabilities. The objective function is convex. Thus, running gradient descent (with or without weight decay) from initialization W_0 leads to maximum minimum. The authors pretend (without more proof details) that it leads to estimates of the following form:

$$\hat{W} = \alpha_0 W_0 + \sum_{i,j} \alpha_{ij} w_U(j) w_E(i)^T$$

They also claim that the impact of W_0 becomes negligible in high dimension. Thus, the learned matrix becomes a combination of the outer products of the input and output embeddings, characterizing the associative memory.

Demonstration:

$$L(W) = \mathbb{E}_{(z,y)\sim p}[\ell(y, W_U W w_E(z))] \ , \quad \text{with } \ell \text{ the cross-entropy loss.}$$

Its derivatives take the form

$$\frac{\partial \ell}{\partial \xi_k}(y, \xi) = \sigma(\xi)_k - \mathbb{1}\{y=k\} \ , \quad \text{with } \sigma(\xi)_k = \frac{e^{\xi_k}}{\sum_j e^{\xi_j}} \text{ the softmax of class k}$$

The gradient of L is then given by

$$\begin{aligned}
\nabla_W L(W) &= \mathbb{E}_{(z,y)}\left[\nabla_W \ell(y, W_U W w_E(z))\right] \\
&= \mathbb{E}_{(z,y)}\left[\sum_{k=1}^{N} \frac{\partial \ell}{\partial \xi_k}(y, W_U W w_E(z)) \nabla_W (w_U(k)^T W w_{\mathrm{E}}(z))\right] \\
&= \mathbb{E}_{(z,y)}\left[\sum_{k=1}^{N} (\hat{p}_W(k|z) - \mathbb{1}\{y=k\}) w_U(k) w_{\mathrm{E}}(z)^T\right] \ , \quad \text{where } \hat{p}_W(k|z) = \sigma(W_U W w_E(z))_k \\
&= \sum_{k=1}^{N} \mathbb{E}_z\left[\mathbb{E}_y[(\hat{p}_W(k|z) - \mathbb{1}\{y=k\}) w_U(k) w_{\mathrm{E}}(z)^T \mid z]\right] \\
&= \sum_{k=1}^{N} \mathbb{E}_z\left[(\hat{p}_W(k|z) - \mathbb{E}_y[\mathbb{1}\{y=k\}|z]) w_U(k) w_{\mathrm{E}}(z)^T\right] \quad \text{(by linearity)}
\end{aligned}$$

which yields the desired result.

3.3 Filtering: associative memory with noisy inputs

Lemma 2: Let p be a data distribution on $(x, y) \in \mathbb{R}^d \times [N]$, and consider the following classification problem, with fixed output embeddings W_U:

$$L(W) = \mathbb{E}_{(x,y)\sim p}[\ell(y, W_U W x)]$$

The gradients take the following form:

$$\nabla_W L(W) = \sum_{k=1}^{N} p(y=k) w_U(k)(\hat{\mu}_k - \mu_k)^T$$

where $\mu_k = \mathbb{E}[x|y=k]$ and $\hat{\mu}_k = \mathbb{E}_x\left[\frac{\hat{p}_W(k|x)x}{p(y=k)}\right]$.

Demonstration:

Doing the same as in the Lemma 1 demonstration, we have:

$$\begin{aligned}
\nabla_W L(W) &= \mathbb{E}_{(x,y)}\left[\sum_{k=1}^{N}(\hat{p}_W(k|x) - \mathbb{1}\{y=k\}) w_U(k) x^T\right] \\
&= \sum_{k=1}^{N} w_U(k)\mathbb{E}_x\left[\hat{p}_W(k|x)x\right]^T - \sum_{k=1}^{N}\mathbb{E}_x\left[\mathbb{1}\{y=k\}w_U(k)\mathbb{E}\left[x|y\right]^T\right] \\
&= \sum_{k=1}^{N} p(y=k) w_U(k)\mathbb{E}_x\left[\frac{\hat{p}_W(k|x)x}{p(y=k)}\right]^T - \sum_{k,j=1}^{N} p(y=k)w_U(k)\mathbb{E}\left[x|y=k\right]^T \\
&= \sum_{k=1}^{N} p(y=k) w_U(k)(\hat{\mu}_k - \mu_k)^T
\end{aligned}$$

which yields the desired result.

Application to noise filtering:

We would like to predict y from $x = w_E(y) + p_t$, where p_t is a positional embedding at a random position t in [T], which we would like to ignore.

We further assume that y is uniformly distributed ($p(y=k) = 1/N$) and consider the matrix obtained after one population gradient step with step-size η starting from an initialization $W_0 = 0$.

We have $\hat{p}_W(k|x) = \sigma(W_U W_0 x) = 1/N$,

Thus, $\hat{\mu}_k = \mathbb{E}_x[x] := \overline{\mu} = 1/N\sum_k w_E(k) + 1/T\sum_t p_t$ and $\mu_k = \mathbb{E}[x|y=k] = w_E(x) + 1/T\sum_t p_t$

So, we get: $W_1 = 0 + \frac{\eta}{N}\sum w_U(k)(w_E(k) - \overline{w}_E)^T$, with $\overline{w}_E := \frac{1}{N}\sum_{k=1}^{N} w_E(k)$

When d is large enough to ensure near-orthonormal embeddings, we have for any y and t:

$$W_1(w_E(y) + p_t) \approx \frac{\eta}{N} w_U(y) + O\left(\frac{1}{N^2}\right)$$

This implies:

$$w_U(k)^T W_1(w_E(y) + p_t) \approx \frac{\eta}{N}\mathbb{1}\{k=y\} + O\left(\frac{1}{N^2}\right)$$

The classifier $\hat{y} = \text{argmax}_k w_U(k)^T W_1(w_E(y) + p_t)$ then has essentially perfect accuracy with one gradient step with infinite data (and high dimension d).

4 Experiments

All the code for our experiments can be found on our GitHub : https://github.com/gabriellecaillaud/TDL_BirthTransformer. In our experiments, we chose to illustrate the Lemma 2 and its application to noise filtering,as well as how the different layers of the model learns.

Focus on the attention scores. A metric that is used in the model is the attention score, that can be computed at each layer by $attention\_scores = Q \cdot K^T$. For each query, the token with the largest attention score is selected, via an argmax. Then this selected token is compared to the expected next token. Using that method, we can compute the fraction of correctly aligned queries and keys. This enable us to identify whether the attention mechanism correctly identifies the token that should receive the highest attention for each query. The accuracy score is useful to understand how the model processes the tokens.

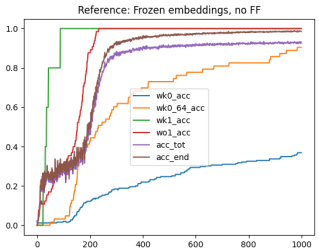
Results We display six curves, which were also featured in the original paper. The layer-specific accuracies correspond to the accuracy explained in the above paragraph.

1. **wk0_acc**. This evaluates how well queries and keys at layer 0 align to predict the next token position.
2. **wk0_64_acc**. This evaluates the alignment between queries and keys at layer 0, but only takes into account the first 64 elements of the sequence.
3. **wk1_acc**. Similar to *wk0_acc*, but for the second attention layer.
4. **wo1_acc**. This indicates if the output projection of the last layer is able to correctly map the attention output back to the original tokens. This accuracy also indicates how much the attention mechanism of layer 2 is able to effectively capture the relationships between tokens and use this information to predict the next token.
5. **acc_tot**. This measures the overall accuracy of token predictions on sequences of length 1 or higher.
6. **acc_end**. This measures the accuracy of token predictions for sequences of length 2 and higher. Since for an input sequence of length n, n next token predictions are made (one for each next token), we are here only taking into account the 500 last predicted tokens of the sequence.

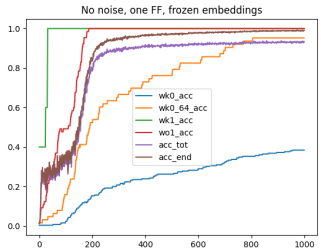
The figures 3(a) and 3(b) are our reference figures: we do not add noise to the embeddings.
The figure 3(c) used the same model as 3(a), except that noise is added. The first four curves of 3(c), which display the attention scores for different layers, are quite the same, but we see that the model takes more time when there is noise to achieve a similar accuracy score for the $wo1$ layer (last projection layer). In addition, while the *acc_end* curve seemed to converge to 1 in 3(a), the same curve converges to 0.9 in 3(c). Even when we add more training steps, as it is done in figure 3(d), the accuracy still converges to 0.9 . As such, we have empirically demonstrated that the model's accuracy regarding the next token prediction drops by around 10% when adding noise to the embeddings. Only using one gradient descent step with infinite data, as introduced in the paper to deal with noisy inputs, is not feasible in practice. Training for more epochs with new data at each epoch is the only way to work around this. The paper's theory is not applicable in real-life transformer training, but our experiment is a close approximation. We show that while

the theory state that the model still reaches perfect accuracy even with noisy inputs, in practice we loose about 10% in accuracy.

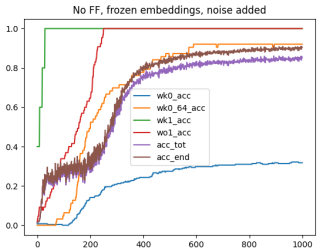
The original article used an embedding layer that is frozen after its random initialization, which enables the embeddings to be semi-orthonormal in practice. The whole theory presented in the paper relies on this orthonormality assumption. However, in real-life transformers, the token embedding layer is learnt during training, so the theory might not hold. We wanted to analyze that in our experiments. To do so, we compare the figures 3(c) and 3(f) : while there are no major differences between the two, the 3(f) with trainable embeddings presents a acc_end accuracy that is a bit higher. The attention score wk1_acc has the exact same shape for both figures: this is because the key linear layer at the second attention head learns the token-alignment early on during training, and the embedding layer might take more time to be learned. Adding a feed-forward layer at the end of the model produce the results shown in 3(e). We see that the only difference with 3(f) is that the projection layer of the second block captures the relationships between the token more quickly.



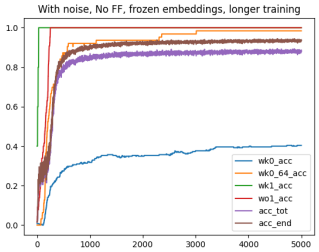
(a) Reference 1: No noise, embeddings are frozen after random initialization, no feed-forward layer after the attention head.



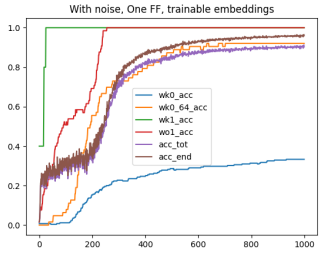
(b) Reference 2: No noise, embeddings are frozen after random initialization, one feed-forward layer after the second attention head.



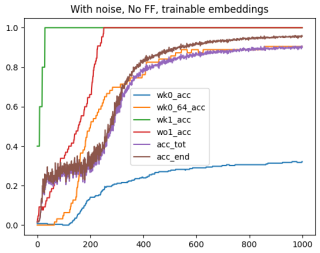
(c) Noise added at the embedding-level. Embedding layer is frozen after random initialization. No Feed-forward layer in the model.



(d) Same model as in subfigure c, but training is five times longer.



(e) With noise added at the embedding level. The embedding layer is not frozen and therefore trained. There is one feed-forward layer after the second attention head.



(f) With noise added at the embedding level. The embedding layer is not frozen and therefore trained. There is no Feed-forward layer in the model.

Figure 3: Experiments that display how the model reacts to noise. This is an illustration of Lemma 2. All hyperparameters are the same for the six experiments, if they are not explicitely described in the subcaptions. We used the same curves as in the paper: the accuracy

5 Discussion

In the article, the authors use a simplified model that do not actually match the real world architectures. As we have detailed in the previous section, this is still useful to understand how some transformer works. However, while in the paper the authors focus on two-layers transformers, real-life transformers often features more blocks (12 for GPT-2). This additional depth is likely to create dynamics in the learning process that have not been covered by the paper. The associative memory presented in the article might not still hold for such transformers. Moreover, all experiments have been done using a synthetically generated dataset, with 5 trigger tokens that are always followed by the same tokens. In a real life dataset, each token can be trigger token, and the specificity of a real life language is that a same word can have different meanings, depending on the context. A real-life languages cannot be modeled by a bigram dataset, where only connections between two consecutive tokens matter.

References

[1] Alberto Bietti et al. *Birth of a Transformer: A Memory Viewpoint*. 2023. arXiv: 2306.00802 [stat.ML]. URL: https://arxiv.org/abs/2306.00802.

[2] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[3] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165.