Theory for Understanding Transformers: An Overview of Current Research

Honam Wong [1] Yicheng Wang [1] Chinghei Mok [1] Yuhao Zhang [1]

Abstract

Transformers are widely used in machine learning and have played a pivotal role in driving advancements across various domains. This project aims to explore the current theoretical research on Transformers and gain a deeper understanding of the reasons behind their success. We begin by reviewing the current work analyzing the standard Transformer architecture, focusing on training dynamics and expressiveness. Then, we move on to discuss the theoretical analysis of Transformers applied to other areas, such as Computer Vision and Graph, explaining why they continue to succeed in learning such specific structures. Lastly, we introduce theoretical work on in-context learning and some work on building white-box transformer. We hope our survey can inspire further research to understand the practical success of Transformers.

1. Introduction

Transformer-based models have been widely applied in diverse applications such as natural language processing and computer vision, showcasing state-of-the-art performance (Vaswani et al., 2017b; Dosovitskiy et al., 2021). While these practical successes are evident and many empirical explorations are present (Child et al., 2019; Dehghani et al., 2018; Kitaev et al., 2020), it is interesting to further investigate the theoretical aspects of transformers to comprehend their capabilities fully.

The report structure is as follows. First, we revisit and define the standard structure of transformer in a mathematical way (check Appendix A), and then introduce the several pathways to understand it theoretically, i.e. optimization perspective, expressiveness perspective, then we give brief introduction to how Transformer is applied to Computer Vision, Graph, and in-context learning, and how and why the transformer can still achieve phenomenal success in these fields. Lastly, we introduce Yi Ma's work on following Transformer's architecture to develop a white-box transformer where every step inside is mathematically understandable and interpretable(Yu et al., 2023).

[1]Department of Computer Science, The Hong Kong University of Science and Technology. Correspondence to: Honam Wong <hnwongaf@connect.ust.hk>.

2. Optimization

We can analyze how Transformer learns from the optimization prospective, which means fine-grained analysis of the training process. Due to difficulty in analyzing dynamics of compositions of multiple non-linear layers, many current work simplifies the setting to one layer only (Tian et al., 2023a; Fu et al., 2023). The work (Tian et al., 2023a) models the dynamics of the optimizaiton process using differential equations, and discovers the scan-and-snap phenomenon (which is also verified in the experiment), in which the attention weight gets sparser in a $O(\log n)$ rate and after a certain period, the weight looks freezed as the rate drops to $O(\log \log n)$ (check Appendix B for details). The main drawback of this paper is that many unrealistic assumptions are imposed such as no residual connections, infinite token length etc. to simplify the analysis. Recently, their follow-up work (Tian et al., 2023b) which extends the single-layer analysis to multiple layers and avoid many unrealistic assumptions.

Another lines of work analyzes in terms of the optimization geometry (Tarzanagh et al., 2023b;a), they show gradient descent converges in direction to a max-margin solution that separates locally-optimal tokens from non-optimal ones. This helps further investigation into Transformer successfully learns through the lens of optimization.

3. Expressiveness

While transformers perform very well in many *designed* scenarios, for example machine translation, we further want to know its true capability. Similar to the Universal Approximation Theorem on neural networks, we want to know to what extend is theoretically doable to a transformer model. There are many scales to measure the learning ability of transformers, and we have chosen the following scales:

1. Ability to approximate arbitrary sequence-to-sequence functions
2. Turing completeness
3. *Mechanistic Interpretability*

The significance of this is that we will know the limits of transformers and thus design new models to get around the limitations. Also note that *mechanistic interpretability* is a slightly different measure, which will be explained in its section.

3.1. Universal Approximation

We see in Proposition C.3 that transformers without positional encoding are "permutation-equivariant", i.e. the permutation of the input does not affect the output. The question is then, is it possible to simulate such kind of sequence-to-sequence functions to arbitrary precision? As it turns out, by (Yun et al., 2020), it is possible with just 2 heads of size 1, and a feed-forward layer with 4 hidden nodes under the L^p norm. The way to do this is as follows

1. Approximate f with piece-wise constant functions
2. Approximate the piece-wise functions with modified transformers
3. Approximate the modified transformer with actual transformers

If we also account for transformers with positional encoding, we can also approximate sequence-to-sequence functions that are not permutation-equivariant using a similar method and contextual mappings. However, it is restricted to continuous functions defined on compact domains. For details, please see Appendix C.2.

3.2. Turing Completeness

The proof of transformers' Turing completeness revolves around construction of a transformer that can simulate a Turing machine. As a reminder, denote Q has the set of all possible states, Σ as the alphabet, a Turing machine $M = (\delta, q, F)$ consists of

1. An internal state $q \in Q$
2. A set of final states $F \subset Q$ which the machine halts when reached
3. A transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\text{left}, \text{right}\}$ that tells the machine what to write and where to move on each step

And an input tape $S \in \Sigma^*$ on which the machine work on.

In order to be Turing complete, we need the transformer to be able to simulate δ and store the states somewhere. Luckily, in a transformer, the output dimension does not need to be the same as the input dimension, and if we restrict the type of programs to ones that halt in a finite amount of time, it is possible to use each output dimension as one step of the program.

As described by (Pérez et al., 2021), it is possible to simulate any Turing machine using Transformers with

1. 1 encoder layer
2. 3 decoder layers
3. Hard attention $(-|\langle \vec{u}, \vec{v} \rangle|)$
4. Embedding space of dimension $2|Q| + 4|\Sigma| + 11$

The construction focuses heavily on the decoder (please refer to Appendix C) since it can output arbitrary length (see Figure 1).

3.3. Mechanical Interpretability

Mechanical Interpretability differs from *expressiveness* in that instead of measuring what a model can do, we measure whether a model *understands* what it is doing. Please refer to Appendix C.3.

4. Transformer applied to Other Areas

The standard transformer model has been incredibly successful in the field of natural language processing. However, it is intriguing to understand how this model is able to capture patterns in diverse types of data, including images and graphs. In the following sections, we will explore additional research that aims to characterize its capacity to learn these specific data forms.

4.1. Computer Vision

ViT (Dosovitskiy et al., 2021) is similar to Transformer but it splits image into fixed-size patches and feed as tokens (Architecture illustrated in Figure 2), and its performance surpasses CNN in major experiments. But why does it achieve superior performance even though Transformer architecture is not originally designed for Computer Vision tasks? While there exists some empirical work trying to explore the reason behind (Raghu et al., 2021; Melas-Kyriazi, 2021; Trockman & Kolter, 2023), there also exists other theoretical work trying to characterize such phenomenon. (Li et al., 2023a) finds that Vision Transformer specifically learns spatially localized patterns (like CNN), and (Jelassi et al., 2022) theoretically analyzes Vision Transformer with

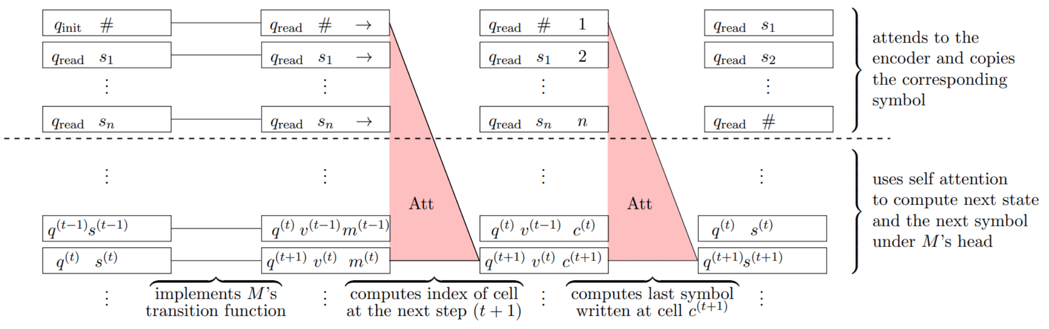

attends to the encoder and copies the corresponding symbol
uses self attention to compute next state and the next symbol under M's head
Att
implements M's transition function
computes index of cell at the next step $(t+1)$
computes last symbol written at cell $c^{(t+1)}$

Figure 1. Visualization of the construction by (Pérez et al., 2021).

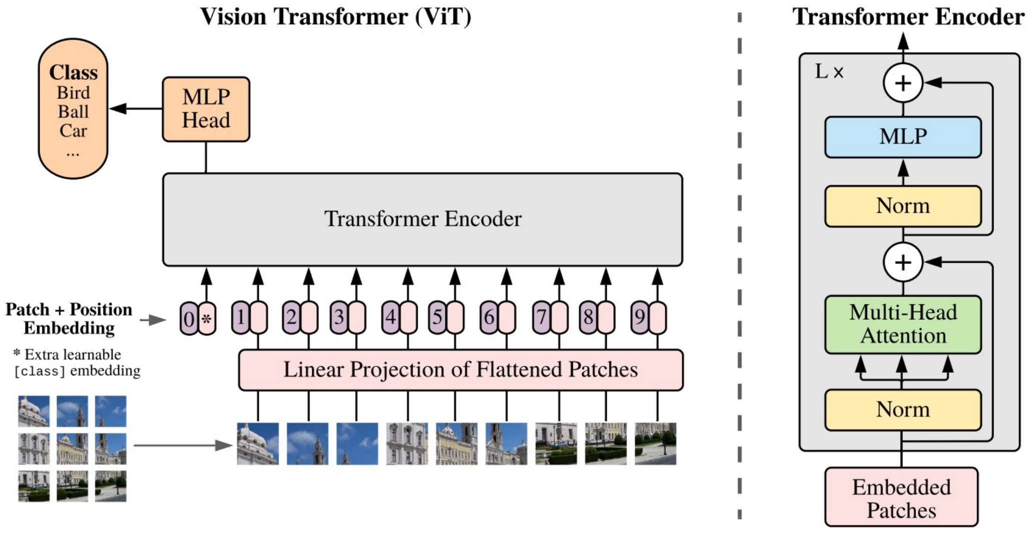


Figure 2. ViT Architecture. Credit to the authors (Dosovitskiy et al., 2021).

one-layer attention and gives Sample Complexity, which is positively correlated with the inverse of the fraction of label-relevant tokens, the token noise level, and the initial model error.

4.2. Graph

There have been many works that adopt Transformers or the attention mechanism to message passing neural networks (MPNN) (Gilmer et al., 2017), for example, Graph Attention Network (Veličković et al., 2018). These models usually incorporate attention signal into the message passing process.

However, such methods may have the following limitations:

1. May inherit limitations in expressiveness of MPNNs.
2. Cannot fully utilized techniques developed for standard Transformers.
3. Desgined for graph specificity and may not be that general.

The TokenGT paper (Kim et al., 2022) discussed the possibility of using pure transformer as an expressive encoding model for graph data.

Given a graph with features on nodes and edges, the goal is to predict certain graph-level feature representation.

TokenGT aims to come up with a tokenization method which takes in a raw graph and outputs a sequence of tokens preserving the structure information. Note that, unlike the positional encoding separated from word embeddings for natural language, TokenGT incorporates the structure information of the graph (not exactly equivalent to position encodings) into the tokenization process and feeds the token sequence into a standard Transformer. The architecture is illustrated in Figure 3 and will be detailed in Appendix D.1.1.

Theorem 4.1 (Corollary 2)**.** *TokenGT (a Transformer on node and type identifiers with a specific configuration) is at*

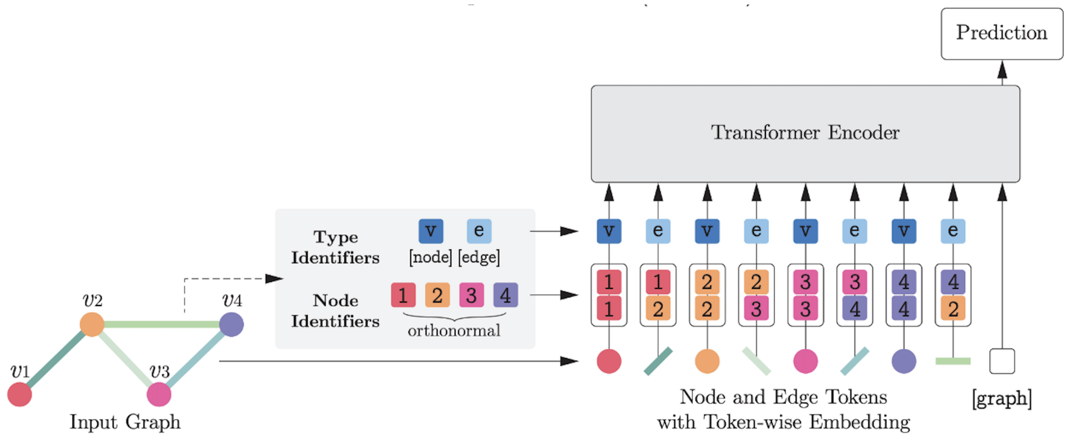


Figure 3. TokenGT Architecture. Credit to the authors (Kim et al., 2022).

least as powerful as k-WL graph isomorphism test and is more expressive than all message-passing GNNs.

To prove this result, the paper first show that pure self-attention on the specifically designed tokens can accurately approximate any equivariant linear basis operator (Maron et al., 2019) on graphs. This leads to the comparison of expressiveness between TokenGT and WL test. Additionally, the paper extend this to order-k TokenGT for hypergraphs.

4.3. Language Model

As we know, GPT models are still based on Transformers (Brown et al., 2020), research analyze its theoretical capacity by analyzing its performance its performance in in-context learning (Dong et al., 2023). (Dai et al., 2023) shows it implicitly performs gradient descent and (Bai et al., 2023) shows Transformer can implicitly perform some statistical algorithms like least squares, ridge regression, Lasso, learning generalized linear models.

There also exists an interesting work which analyzes Chain of Thought theoretically by modelling the tasks into Arithmetic, Equation Solving, and Dynamic Programming, and applying circuit complexity theory to model and explain effectiveness of CoT (Feng et al., 2023).

5. White-box Transformer

Transformers (Vaswani et al., 2017a) is notorious for being black-box as we could not fully understand what it does inside. To address this issue, Ma Yi proposed a White-box transformer-like architecture known as CRATE (Yu et al., 2023), which aims to produce fully mathematically interpretable representations throughout the learning process.

As a brief introduction (details refer to Appendix E), this architecture provides iterative unrolling optimization schemes to optimize the sparse rate reduction objective:

$$\arg\min_{f\in F} E_Z[\underbrace{R^c(Z;U_{[K]})}_{\text{compression}} + \underbrace{\|Z\|_0 - R(Z)}_{\text{sparsification}}]$$

The first compression term is to minimize the inter-class distances while the second sparsification term is to maximize the intra-class distances in the dataset. This may seem like representation learning (Bengio et al., 2013) but with the class label.

The experiment shows that CRATE can learn the desired compressed and sparse representations on large-scale real-world datasets and achieve a good performance in various tasks.

Despite its potential, the CRATE transformer still faces certain challenges. In experimental comparisons, it failed to outperform the ViT-S model (Arnab et al., 2021). Additionally, the implementation of the CRATE transformer did not align well with the theoretical concepts presented in the paper, raising concerns about its practical applicability.

In summary, while the CRATE transformer offers a promising solution to the lack of interpretability in transformers, its current limitations hinder its effectiveness. Further research and improvements are necessary to enhance its performance.

6. Summary

We have surveyed various directions into understanding Transformers theoretically, including analyzing its training process, and its expressiveness. We also examine other theoretical analysis into Transformers applied to other forms of data. In the end, we examine Yi Ma's attempt on designing a mathematically interpretable transformer-like architecture.

We hope our overview can provide guidance to people interested in doing research on investigating reason behind the success of Transformer.

References

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6836–6846, 2021.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Chan, K. H. R., Yu, Y., You, C., Qi, H., Wright, J., and Ma, Y. Redunet: A white-box deep network from the principle of maximizing rate reduction. *The Journal of Machine Learning Research*, 23(1):4907–5009, 2022.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Dai, D., Sun, Y., Dong, L., Hao, Y., Ma, S., Sui, Z., and Wei, F. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers, 2023.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., Li, L., and Sui, Z. A survey on in-context learning, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Feng, G., Zhang, B., Gu, Y., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: A theoretical perspective, 2023.

Fu, H., Guo, T., Bai, Y., and Mei, S. What can a single attention layer learn? a study through the random features lens, 2023.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Thirty-fourth International Conference on Machine Learning*, 2017.

Jelassi, S., Sander, M., and Li, Y. Vision transformers provably learn spatial structure. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 37822–37836. Curran Associates, Inc., 2022.

Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 14582–14595. Curran Associates, Inc., 2022.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Li, H., Wang, M., Liu, S., and Chen, P.-Y. A theoretical understanding of shallow vision transformers: Learning, generalization, and sample complexity. In *The Eleventh International Conference on Learning Representations*, 2023a.

Li, S., Song, Z., Xia, Y., Yu, T., and Zhou, T. The closeness of in-context learning and weight shifting for softmax regression, 2023b.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

Melas-Kyriazi, L. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet, 2021.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Pérez, J., Barceló, P., and Marinkovic, J. Attention is turing-complete. *Journal of Machine Learning Research*, 22 (75):1–35, 2021.

Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., and Dosovitskiy, A. Do vision transformers see like convolutional neural networks? In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021.

Tarzanagh, D. A., Li, Y., Thrampoulidis, C., and Oymak, S. Transformers as support vector machines, 2023a.

Tarzanagh, D. A., Li, Y., Zhang, X., and Oymak, S. Max-margin token selection in attention mechanism, 2023b.

Tian, Y., Wang, Y., Chen, B., and Du, S. Scan and snap: Understanding training dynamics and token composition in 1-layer transformer. 2023a.

Tian, Y., Wang, Y., Zhang, Z., Chen, B., and Du, S. Joma: Demystifying multilayer transformers via joint dynamics of mlp and attention. 2023b.

Trockman, A. and Kolter, J. Z. Patches are all you need? *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017a.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017b.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Yu, Y., Chu, T., Tong, S., Wu, Z., Pai, D., Buchanan, S., and Ma, Y. Emergence of segmentation with minimalistic white-box transformers. *arXiv preprint arXiv:2308.16271*, 2023.

Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S., and Kumar, S. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020.

A. Standard Transformer Structure

This section serves as a very abstract description of the Transformer model. We first define some notations used

1. V, an embedding space where each vector inside represent some kind of meaning, and $\dim V = d$
2. Σ_I and Σ_O, the alphabet of the input and output
3. $S \in \Sigma_I^*$, the input to the transformer
4. $S' \in \Sigma_O^*$, the output of the transformer
5. $X = (\vec{x}_1, \ldots, \vec{x}_n)$ where $\vec{x}_i \in V$, n is fixed
6. $Z = (\vec{z}_1, \ldots, \vec{z}_n)$ where $\vec{z}_i \in V$
7. $Y = (\vec{y}_1, \ldots, \vec{y}_m)$ where $\vec{y}_i \in V$, m is dynamic, usually controlled by a seed

The rough idea of a transformer is a chain of maps in the following fashion

$$S \xmapsto{\mathit{Embedder}} X \xmapsto{\mathit{Encoder}} Z \xmapsto{\mathit{Decoder}} Y \xmapsto{\mathit{Embedder}^{-1}} S'$$

A.1. Encoder ($X \mapsto Z$)

Essentially what an encoder does is extract the meaning of its input X by performing some linear transformation on X, and give it to the decoder for generating the output. The way a transformer does this is by passing X though h different trained linear maps that we call "attention" *simultaneously*, then return their "sum".

$$\mathit{Encoder} : X \xmapsto{\mathit{Attention}} \{A_i\}_{i=1}^h \xmapsto{\sum} Z$$

In production, we chain ℓ different encoder layers together and use the return value of the last one as Z.

$$X \xmapsto{\mathit{Encoder}_1} X_1 \xmapsto{\mathit{Encoder}_2} \cdots \xmapsto{\mathit{Encoder}_{\ell-1}} X_{\ell-1} \xmapsto{\mathit{Encoder}_\ell} Z$$

It is worth noting that X, X_i and Z all belong to V^n, so $\mathit{Encoder}_i \in \mathrm{End}(V^n)$.

A.2. Decoder ($Z \mapsto Y$)

Similar to the encoder, the decoder also maps tuples in V to tuples in V, but the dimension of input and output can be different. It also accept the output from the encoder Z as parameters. And similar to the encoder, the decoder also has an attention mechanism.

$$\mathit{Decoder} : Z \xmapsto{\mathit{Attention}} \{B_i\}_{i=1}^h \xmapsto{\sum} Z' \xmapsto{\mathit{Attention}+Z} \{C_i\}_{i=1}^h \xmapsto{\sum} Y$$

Similar to the encoder, we can also chain up multiple decoder layers

$$Z \xmapsto{\mathit{Decoder}_1} Z_1 \xmapsto{\mathit{Decoder}_2} \cdots \xmapsto{\mathit{Decoder}_{\ell-1}} Z_{\ell-1} \xmapsto{\mathit{Decoder}_\ell} Y$$

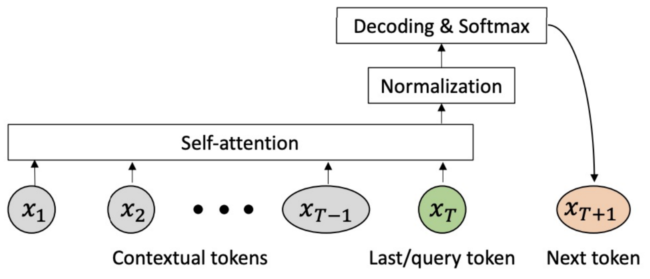
A.3. Attention

The attention mechanism is the core of Transformers. The attention weights are computed using a mechanism known as the scaled dot-product attention. The input X is mapped into a query vector Q, a set of key vectors $K = (\vec{k}_1, \vec{k}_2, \ldots, \vec{k}_n)$ and a set of value vectors $V = (\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n)$, and then concatenated back into a tuple of vectors in the embedding space V, essentially extracting information from the input, refine it, and put it back.

$$\mathit{Attention} : X \xmapsto{\langle,\rangle} (Q, K, V) \xmapsto{\sum} A$$

B. Fined-grained Analysis of the training process

We give more details on explaining the paper (Tian et al., 2023a) here, a one-layer setting is considered for simplifying the analysis as follows, where $U = [u_1, u_2, \cdots u_M]^T$ is the token embedding matrix, $\hat{u}_T = \sum_{t=1}^{T-1} b_{tT} u_{xt} = U^T X^T b_T$, where b_{tT} is defined as $\frac{\exp(u_{xT} W_Q W_K^T u_{x_t}/\sqrt{d})}{\sum_{t=1}^{T-1} \exp(u_{xT} W_Q W_K^T u_{x_t}/\sqrt{d})}$, then the normalized version $\tilde{u}_T = U^T LN(X^T b_T)$, the objective is the max entropy loss i.e. $\max_{W_K, W_Q, W_V, U} J = \mathbb{E}_D[u_{x_{T+1}}^T W_V \tilde{u}_T - \log \sum_l \exp(u_l^T W_V \tilde{u}_T)]$.



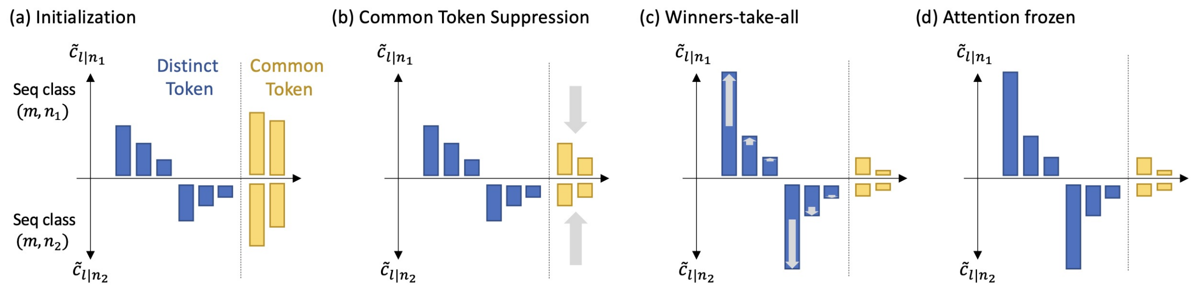
The author parameterize the two layers, namely decoder $Y = UW_V^T U^T$,and self-attention layer $Z = UW_Q W_K^T U^T$, similar parametrization also appears in other work (Jelassi et al., 2022), (Li et al., 2023b). Then, the author derives the training dynamics of Y and Z as a differential equation.

Theorem B.1 (Dynamics of Decoder and Attention layer)**.**

$$\dot{Y} = \eta_Y \, \mathrm{LN}\left(X^\top \boldsymbol{b}_T\right)\left(\boldsymbol{x}_{T+1} - \boldsymbol{\alpha}\right)^\top, \quad \dot{Z} = \eta_Z \boldsymbol{x}_T \left(\boldsymbol{x}_{T+1} - \boldsymbol{\alpha}\right)^\top Y^\top \frac{P^\perp_{X^\top \boldsymbol{b}_T}}{\left\|X^\top \boldsymbol{b}_T\right\|_2} X^\top \operatorname{diag}\left(\boldsymbol{b}_T\right) X$$

Note that several assumptions are given to facilitate the analysis i.e.

1. No positional encoding and residual connections
2. Sequence length $T \to \infty$
3. Learning rate of decoder Y larger than self-attention layer Z ($\eta_Y >> \eta_Z$)
4. Other technical assumptions



The paper characterizes distinct and common tokens formally in the concept of sequence classes. At initialization, the attention logit is 0, then $\dot{z_m l} < 0$ for common token l, and > 0 for distinct token l, which leads to the "suppression" of common tokens (Theorem 2). Note that $z_{ml}(t)$ grows faster with larger $\mathbb{P}(l|m,n)$, which leads to winner-emergence – the rapid increase of contextual sparsity, the rate is bounded given in Theorem 3 of the paper i.e:

Theorem B.2 (Relative Gain)**.** *Relative gain* $r_{l/l'|n}(t) := \frac{\tilde{c}^2_{l|n}(t)}{\tilde{c}^2_{l'|n}(t)}$ *has a closed form:*

$$r_{l/l'|n}(t) = r_{l/l'|n}(t)\chi_l(t)$$

If l_0 is the dominant token, $r_{l_0/l|n}(0) > 0$ for all $l \neq l_0$ then

$$e^{2f_{nl_0}^2 B_n(t)} \le \chi_{l_0}(t) \le e^{2B_n(t)}$$

where $B_n(t) \geq 0$ monotonously increases, and $B_n(0) = 0$.

Further, $B_n(t)$'s rate is given and has different properties before and after a state transition as follows:

Theorem B.3 (Two stages)**.** *When $t \to \infty$,*

$$B_n(t) \sim \ln(C_0 + 2K\frac{\eta_Z}{\eta_Y}\ln^2(\frac{M\eta_Y t}{K}))$$

There are two stages i.e.

Attention Scanning: *When training starts, the attention weight gets sparser, $B_n(t) = O(\ln t)$.*

Attention Snapping: *When $t \geq t_0 = O(\frac{2K \ln M}{\eta_Y})$, $B_n(t) = O(\ln\ln t)$, the rate gets much smaller, and this remarks the final stage where the attention appears to frozen.*

Empirically, experiment in the paper does demonstrate that attention weight is getting sparser in both the 1-layer and 3-layer setting.

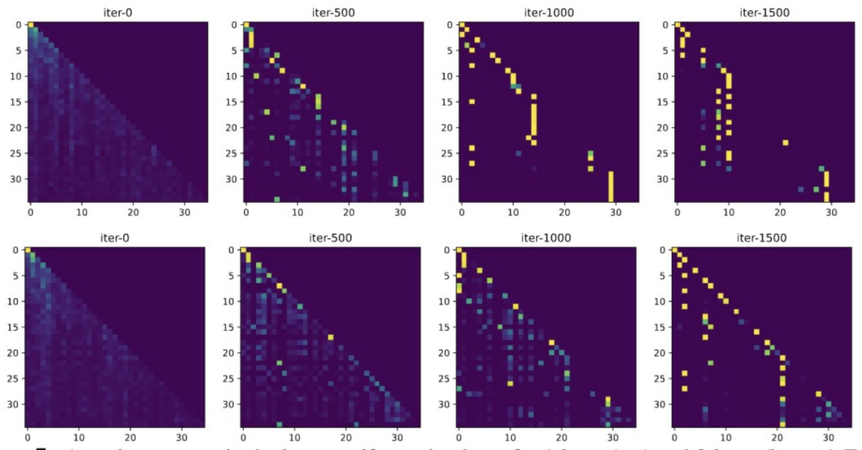


Figure 7: Attention patterns in the lowest self-attention layer for 1-layer (top) and 3-layer (bottom) Transformer trained on WikiText2 using SGD (learning rate is 5). Attention becomes sparse over training.

C. Expressiveness of Transformer

C.1. Turing Completeness

The following section follows the formulation, with some simplification, used in (Pérez et al., 2021). We shall start with the definitions used.

Definition C.1. For a string $w \in \Sigma_I^*$ and a set of desired vectors $\mathbb{F} \subset V$ is accepted by a sequence-to-sequence function N if $N(w) = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_m)$ and $\vec{y}_m \in \mathbb{F}$.

Definition C.2. The set of all strings accepted by a sequence-to-sequence function N is called its language, denoted as $\mathcal{L}_N$.

Note that the definition of being accepted only depends on the last output vector $\vec{y}_m$. The motivation behind this is that we try to mimic the definition of a Turing machine, i.e. treating the output m-tuple as the history of states, and the last one as the final state. We want to find a *family* $\mathcal{N}$ of sequence-to-sequence functions that, together, contains all the strings accepted by a Turing machine.

C.1.1. Positional Encoding

A transformer, without positional encoding, is almost a function that is invariant on permutations of the input.

Proposition C.3. *We define*

1. *The function prop(a, w) is the ratio of $a \in \Sigma_I$ in $w \in \Sigma_I^*$*
2. *The function PropInv$(w) = \{u \in \Sigma_I^* :$ prop$(a, w) =$ prop(a, u) for all $a \in \Sigma_I\}$, which is the set of all strings that is a permutation of strings with same alphabet ratio as w*

It follows that $u \in$ PropInv$(w) \implies T(w) = T(u)$ where T is a transformer ***without*** *positional encoding.*

This property is followed by

1. The order-invariant regular language $\mathcal{L} = \{w \in \{a, b\}^* : w$ has even number of $a\}$ cannot be accepted by a transformer
2. The non-regular language $\mathcal{S} = \{w \in \{a, b\}^* : w$ has strictly more a than $b\}$ can be accepted by a transformer

Therefore it is necessary that we include positional encoding in the model in order for it to be Turing complete.

C.1.2. Proof of Turing Completeness

Let M be a Turing machine we want to simulate, denote

1. $q^{(i)}$ be the state of M at step i
2. $s^{(i)}$ be the symbol read at step i
3. $v^{(i)}$ be the symbol written to at step i
4. $m^{(i)}$ be the direction in which the head moves during step i

Then $\delta(q^{(i)}, s^{(i)}) = (q^{(i+1)}, v^{(i)}, m^{(i)})$ represents the transition done by step i, and if we can simulate this function with a transformer, we are done.

For the encoder, we try to construct such that

1. Attention: score$(\vec{u}, \vec{v}) = -|\langle \vec{u}, \vec{v} \rangle|$
2. Embedding space V with size $2|Q| + 4|\Sigma| + 11$ that looks something like this for $\vec{v} \in V$:

$$\vec{v} = (\vec{q}_1, \vec{s}_1, x_1, \quad \vec{q}_2, \vec{s}_2, x_2, \ldots, x_5, \quad \vec{s}_3, x_6, \vec{s}_4, x_7, \quad x_8, \ldots, x_{11})$$

3. Embedding function $f(s) = (0, \ldots, 0, \quad 0, \ldots, 0, \quad s, 0, \vec{0}, 0, \quad 0, \ldots, 0)$

4. Positional encoder: $\text{pos}(i) = (0, \ldots, 0, \quad 0, \ldots, 0, \quad 0, \ldots, 0, \quad 1, i, \frac{1}{i}, \frac{1}{i^2})$

5. Encoder (K^e, V^e) where $K^e = (\vec{k}_1, \ldots, \vec{k}_n)$ and $V^e = (\vec{v}_1, \ldots, \vec{v}_n)$, and

   (a) $\vec{k}_i = (0, \ldots, 0, \quad 0, \ldots, 0, \quad 0, \ldots, 0, \quad i, -1, 0, 0)$

   (b) $\vec{v}_i = (0, \ldots, 0, \quad 0, \ldots, 0, \quad \vec{s}_i, i, \vec{0}, 0, \quad 0, \ldots, 0)$

Lemma C.4. *Let $\vec{q} \in V$ such that $\vec{q} = (\ldots, \quad \ldots, \quad 1, j, \ldots, \quad \ldots)$, where $\ldots$ means the value is arbitrary,*

$$\mathit{Att}(\vec{q}) = (0, \ldots, 0, \quad 0, \ldots, 0, \quad \alpha^{(j)}, \beta^{(j)}, \vec{0}, 0, \quad 0, \ldots, 0)$$

where $\beta^{(i)} = \min\{i, n\}$ and $\alpha^{(i)} = s_{\beta^{(i)}}$.

For the decoder, we desire the output $Y = (\vec{y}_1, \ldots, \vec{y}_m)$ be

$$V \ni \vec{y}_i = \left(q^{(i)}, s^{(i)}, m^{(i-1)}, \quad 0, \ldots, 0, \quad 0, \ldots, 0, \quad 1, (i+1), \frac{1}{i+1}, \frac{1}{(i+1)^2} \right)$$

Further, we give it a seed vector $\vec{z}_0 = (q_{init}, \#, 0, \quad 0, \ldots, 0, \quad 0, \ldots, 0, \quad 0, \ldots, 0)$, similar to initializing a Turing machine.

Lemma C.5. *There exists a 3-layer decoder which does the following:*

$$\mathit{Att}(\vec{y}_i, K^e, V^e) + \vec{y}_i = \left(q^{(i)}, s^{(i)}, m^{(i-1)}, \quad 0, \ldots, 0, \quad \alpha^{(i+1)}, \beta^{(i+1)}, \vec{0}, 0, \quad 1, (i+1), \frac{1}{i+1}, \frac{1}{(i+1)^2} \right)$$
$$\mapsto \left(0, \ldots, 0, \quad q^{(i+1)}, v^{(i+1)}, m^{(i)}, m^{(i-1)}, 0, 0, \quad \alpha^{(i+1)}, \beta^{(i+1)}, \vec{0}, 0, \quad 1, (i+1), \frac{1}{i+1}, \frac{1}{(i+1)^2} \right)$$
$$\mapsto \left(0, \ldots, 0, \quad q^{(i+1)}, v^{(i+1)}, m^{(i)}, m^{(i-1)}, \frac{c^{(i+1)}}{i+1}, \frac{c^{(i)}}{i+1}, \quad \alpha^{(i+1)}, \beta^{(i+1)}, \vec{0}, 0, \quad 1, (i+1), \frac{1}{i+1}, \frac{1}{(i+1)^2} \right)$$
$$\mapsto \left(0, \ldots, 0, \quad q^{(i+1)}, v^{(i+1)}, m^{(i)}, m^{(i-1)}, \frac{c^{(i+1)}}{i+1}, \frac{c^{(i)}}{i+1}, \quad \alpha^{(i+1)}, \beta^{(i+1)}, v^{(\ell(i+1))}, \ell(i+1), \quad 1, (i+1), \frac{1}{i+1}, \frac{1}{(i+1)^2} \right)$$
$$\mapsto \left(q^{(i+1)}, s^{(i+1)}, m^{(i)}, \quad 0, \ldots, 0, \quad 0, \ldots, 0, \quad 1, i+2, \frac{1}{i+2}, \frac{1}{(i+2)^2} \right) = \vec{y}_{i+1}$$

where $c^{(i+1)} = \sum_{j=1}^{i} m^{(j)}$, the displacement of the Turing machine head; $\ell(i)$ is the last time when the machine head pointed at $c^{(i)}$. The first 3 $\mapsto$ each is a decoder layer.

Therefore, we can simulate Turing machine, and thus transformers are Turing complete. The detailed constructions can be found in (Pérez et al., 2021).

C.2. Universal Approximators of Sequence-To-Sequence Functions

C.2.1. Formulation

The paper (Yun et al., 2020) formulates the Transformer block as follows, with layer normalization omitted to simply the analysis:

$$\text{Transformer}(\mathbf{X}) = \text{Attn}(\mathbf{X}) + \text{MLP}(\text{Attn}(\mathbf{X}))$$
$$\text{Attn}(\mathbf{X}) = \mathbf{X} + \sum_{i=1}^{h} \mathbf{W}_O^i \mathbf{W}_V^i \mathbf{X} \cdot \sigma[(\mathbf{W}_K^i \mathbf{X})^T \mathbf{W}_Q^i \mathbf{X}],$$
$$\text{MLP}(\text{Attn}(\mathbf{X})) = \mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1 \cdot \text{Attn}(\mathbf{X}) + \mathbf{b_1} \vec{1}_n^T) + \vec{b_2} \vec{1}_n^T.$$

C.2.2. THEORY

Main theorem:

Proposition C.6 (Claim 1)**.** *A Transformer block is a permutation equivariant (PE) map $f : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$, i.e., for any permutation matrix $\mathbf{P}$, we have $f(\mathbf{XP}) = f(\mathbf{X})\mathbf{P}$.*

Note that $\mathbf{X}$ here is the immediate input to the Transformer block. More specifically, let $\mathbf{S}$ be the raw token vectors of the input sentence, and $\mathbf{E}$ be the positional encodings. Then, if $\mathbf{X} = \mathbf{S}$ (without positional encodings), then f is PE in the sentence; if $\mathbf{X} = \mathbf{S} + \mathbf{E}$ (with positional encodings), then f not PE in the sentence but in the resulting $\mathbf{S} + \mathbf{E}$.

In the following analysis, we first investigate the Transformer block f and extend the result to the case in which positional encodings are added in the input.

Theorem C.7 (Theorem 2)**.** *Let $1 \le p < +\infty$. For any continuous PE function with compact support (we denote this class by $\mathcal{F}_{PE}$), there exists a Transformer network that universally approximates it w.r.t. the L^p function norm. Formally, for any $\epsilon > 0$, for any $f \in \mathcal{F}_{PE}$, there exists a Transformer network g such that $\|f - g\|_p \le \epsilon$.*

Then the following theorem further considers the positional encodings.

Theorem C.8 (Theorem 3)**.** *Let $1 \le p < +\infty$.For any continuous map on compact domain (we denote this class by $\mathcal{F}_{CD}$), there exists a Transformer network with positional encodings that universally approximates it. Formally, for any $\epsilon > 0$, for any $f \in \mathcal{F}_{CD}$, there exists a Transformer network with positional encodings g such that $\|f - g\|_p \le \epsilon$.*

C.2.3. PROOF OF THEOREM 2

The proof of Theorem 2 goes in three steps:

1. Approximate $\mathcal{F}_{\text{PE}}$ with PE piece-wise constant functions $\overline{\mathcal{F}}_{\text{PE}}(\delta)$.
2. $\star$ Approximate $\overline{\mathcal{F}}_{\text{PE}}(\delta)$ with modified Transformers.
3. Approximate modified Transformers with (original) Transformers.

We will not show the details of these steps. Instead, we introduce the notion of contextual mapping and a related result that may helpful in understanding how Transformers are bridged with these special functions (step 2).

We first define the PE piece-wise constant function class $\overline{\mathcal{F}}_{\text{PE}}(\delta)$. Let grid $\mathbb{G}_\delta := \{0, \delta, \dots, 1-\delta\}^{d\times n}$. Then

$$\overline{\mathcal{F}}_{\text{PE}}(\delta) := \{f : \mathbf{X} \mapsto \sum_{\mathbf{L}\in\mathbb{G}_\delta} \mathbf{A_L}\mathbf{1}\{\mathbf{X} \in \mathbb{S}_\mathbf{L}\} | f \text{ is PE }, \mathbf{A_L} \in \mathbb{R}^{d\times n}\},$$

where $\mathbb{S}_\mathbf{L}$ is the cube $\in [0,1]^{d\times n}$ of length δ located at $\mathbf{L}$: $\mathbb{S}_\mathbf{L} := \{\mathbf{Y} | \mathbf{L} \le \mathbf{Y} \le \mathbf{L} + \delta \text{ entry-wise}\}$. Intuitively, the grid $\mathbb{G}_\delta$ partitions $[0,1]^{d\times n}$ into cubes of length δ and f assigns a constant to points in each cube.

Definition C.9 (Contextual Mapping)**.** Let $\mathbb{L} \subseteq \mathbb{R}^{d\times n}$ be a finite set. A map $q : \mathbb{L} \to \mathbb{R}^{d\times n}$ is a contextual mapping if:

1. For any $\mathbf{L} \in \mathbb{L}$, the n entries in $q(\mathbf{L})$ are distinct: q differentiates every position in a sentence.
2. For any $\mathbf{L}, \mathbf{L}' \in \mathbb{L}$, $\mathbf{L} \ne \mathbf{L}'$, all entries of $q(\mathbf{L})$ and $q(\mathbf{L}')$ are distinct: A token will have different value in different context.

Lemma C.10 (Lemma 6 (informal))**.** *There exist a function $g_a : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ composed of a certain number of self-attention layers and a vector $\vec{u} \in \mathbb{R}^d$ such that $q(\cdot) := u^T g_a(\cdot) : \tilde{\mathbb{G}}_\delta \to \mathbb{R}^{1\times n}$ is a PE contextual mapping for a subset $\tilde{\mathbb{G}}_\delta \subseteq \mathbb{G}_\delta$ that contains almost all elements of $\mathbb{G}_\delta$.:*

1. *the same*
2. *For any $\mathbf{L}, \mathbf{L}' \in \mathbb{L}$, $\mathbf{L}$ is not a permutation of $\mathbf{L}'$, all entries of $q(\mathbf{L})$ and $q(\mathbf{L}')$ are distinct.*

Finally, a group of feed-forward layers in the modified Transformer network can map elements of the contextual embedding $q(L)$ to the desirable values, i.e., the output of $\overline{f} \in \overline{\mathcal{F}}_{\text{PE}}(\delta)$ on the input $\mathbf{X}$.

C.2.4. Proof of Theorem 3

The proof of Theorem 3 also goes in the similar three steps:

1. Approximate $\mathcal{F}_{\mathrm{CD}}$ with piece-wise constant functions $\overline{\mathcal{F}}_{\mathrm{CD}}(\delta)$ on compact domain.
2. $\star$ Approximate $\overline{\mathcal{F}}_{\mathrm{CD}}(\delta)$ with modified Transformers with positional encodings.
3. Approximate modified Transformers with (original) Transformers with positional encodings.

Here we only show the key construction for step 2. Assume, w.l.o.g., that $\mathbf{X} \in [0,1]^{d \times n}$ and choose

$$\mathbf{E} = \begin{pmatrix} 0 & 1 & \dots & n-1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & n-1 \end{pmatrix}.$$

Then, we want to show that attention layers (followed by $\vec{u}^T(\cdot)$) can implement a contextual mapping on the input $\mathbf{X} = \mathbf{S} + \mathbf{E}$.

C.3. Mechanistic Interpretability of Transformer

Specialised transformers show promising sign of artificial general intelligence, and the question is, can transformers actually "learn" concepts? For example, ChatGPT is trained with text, yet it is able to perform simple addition of numbers. To proof that transformers are capable of learning any concept, we must understand the whole transformer as a function and derive the property using mathematics. However, multi-layer transformers are too complicated to analyse, thus we try doing the following

1. Show that one-layer and two-layer transformers are able to match patterns
2. Find the necessary conditions such that we can extend the result in 1 to larger models
3. Verify the conditions in 2 by experiment

The article by (Elhage et al., 2021) explained in-depth how "0", 1, and 2 layers transformers can be described in terms of linear algebra, which also eased the analysis of mechanistic interpretability as it revealed pattern matching abilities in few-layers transformers. In a subsequent article by (Olsson et al., 2022), the term "induction head" was introduced to describe and explain more abstract pattern matching. Extensive experiments were carried out to verify claims that support the argument.

Since any summaries of the 2 articles will be a gross simplification, we will not repeat what was there. For more details, please do visit (Elhage et al., 2021) and (Olsson et al., 2022) for their brilliant works.

D. Theoretical understanding of other variants of Transformer

D.1. TokenGT

D.1.1. Architecture

TokenGT generates one token (a vector) for every node and edge and the order of the token sequence can be arbitrary. Let V be the vertex set and E be the edge set. Define $n = |V|, m = |E|$. Let $\mathbf{X}^V \in \mathbb{R}^{n \times C}$ be the node feature matrix, where the vector $\vec{X}_v$ is the feature for the node v. Similarly, $\mathbf{X}^E \in \mathbb{R}^{m \times C}$ is the edge feature matrix.

Every token consists of three parts:

1. The given feature vector: $\vec{X}_v$ or $\vec{X}_e$.
2. Structure identifier: supposed to (implicitly) indicate the structural information.
3. Type identifier: tells whether it is a node or an edge.

Structure identifiers: Each node is associated with an identifier vector $\vec{P}_u$ and the identifier vectors for all nodes should be mutually **orthogonal**. The matrix collecting all the identifier vectors is called the identifier matrix $\mathbf{P}$. Then:

- For each node $v \in V$, its structure identifier is $[\vec{P}_u; \vec{P}_u]$.
- For each edge $(u, v) \in E$, its structure identifier is $[\vec{P}_u, \vec{P}_v]$.

The $[;]$ notation above represents concatenation of vectors. Intuitively, the dot product of two structure identifiers reflect the connectivity of the edges / nodes the they represent: Consider an edge (u, v) (assume $u \neq v$) and a node k. $[\mathbf{P}_u; \mathbf{P}_v] \cdot [\mathbf{P}_k; \mathbf{P}_k] = \mathbf{P}_u \cdot \mathbf{P}_k + \mathbf{P}_v \cdot \mathbf{P}_k = 1$ iff $k \in \{u, v\}$ and 0 otherwise. However, it is up to the encoder (which is a standard Transformer in this paper) whether such information can be well captured.

Type identifier: Two distinct and trainable vectors $\vec{E}^V, \vec{E}^E \in \mathbb{R}^{d_e}$.

Then, the tokens are defined by:

- For each node $u \in V$, its token is $[\mathbf{X}_u; \mathbf{P}_u; \mathbf{P}_u; \vec{E}^V]$.
- For each node $(u, v) \in E$, its token is $[\mathbf{X}_{(u,v)}; \mathbf{P}_u; \mathbf{P}_v; \vec{E}^E]$.

For the choices of structure identifiers, the paper experimented with Orthogonal Random Features (ORFs) and Laplacian Eigenvectors (Lap). It claimed that Lap provides a kind of graph **positional information** and performs better compared to ORFs. Lap can be viewed as a generalization of sinusoidal positional embeddings for NLP transformers.

D.1.2. Theory

Here are the building blocks of the theorem.

Definition D.1 (Definition 1)**.** A k-IGN $F_k \in \mathbb{R}^{n^k \times d_0} \to \mathbb{R}$ can be written as a composition equivariant linear layers, activation functions and an MLP:

$$F_k = \text{MLP} \circ L_{k \to 0} \circ L^{(T)}_{k \to k} \circ \sigma \circ \cdots \circ \sigma \circ L^{(1)}_{k \to k},$$

where L represents a equivariant linear layer (see below)

Definition D.2 (Definition 2)**.** An equivariant linear layer can be written as combination of equivariant basis tensors (applied to the input).

From the similarity between the attention mechanism and the equivariant linear layer in the basis tensor form, the paper showed the following:

Proposition D.3 (Lemma 1)**.** *A self-attention can approximate any equivariant basis tensor.*

And this leads to the conclusions of the paper:

Theorem D.4 (Theorem 1)**.** *A transformer layer with certain number (bell-2k) of self-attention heads can approximate an equivariant layer.*

Theorem D.5 (Theorem 2)**.** *A stack of transformer layers (followed by some other functions) can approximate a k-IGN.*

E. Attempt to white-box transformer

We elaborate Yi Ma's line of work here:

E.1. Rethinking purpose of NN: Information Bottleneck Theory

We take the classification problem as an example, machine learning models are trying to do the following two steps:

$$x \xmapsto{f(x,\theta)} z(\theta) \xmapsto{g(z)} y$$

In which $z(\theta)$ means a representation of input x. This formula means that the model uses the encoder f to get a representation of the input, then uses this representation as while as a downstream classifier g to handle the whole process.

According to the information bottleneck(IB) theory, the representation that which model learns is to optimize :

$$max_{\theta \in \Theta} IB(x, y, z(\theta)) = I(z(\theta), y) - \beta I(x, z(\theta)), \beta > 0$$

In the above function, $I(z, y) = H(z) - H(z|y)$ means the mutual information of z, y. $I(z)$ is the Shannon Entropy of z. Thus to optimize the first term means to learn more mutual information between the representation and the output, and the second term means to learn less mutual information between the representation and the input. This means learning features involve selecting input information while simultaneously narrowing the distance between the input and output information.

This theory points out the existing problem of the gradient descent algorithm: the representation learned from data is related to the downstream task, which means the representation may suffer a data loss when the downstream task changes. Which has been proved by the experiments.

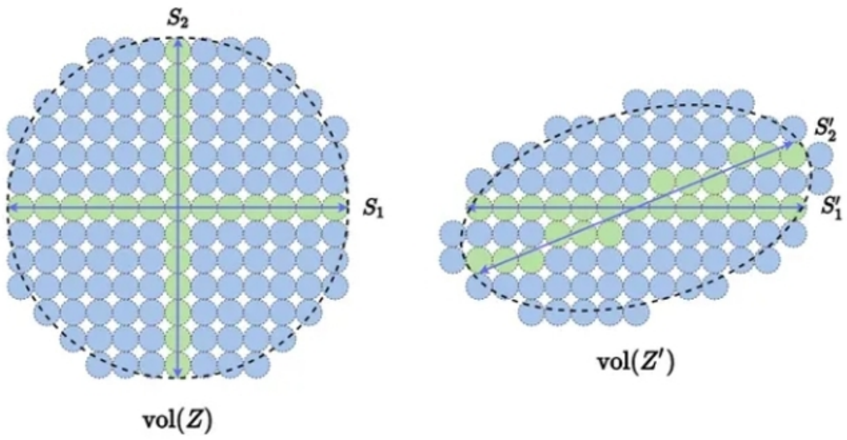
E.2. Objective: Maximal Coding Rate Reduction (Chan et al., 2022)

Classification problems and clustering problems are essentially the same. In a classification problem, for each data point, the model extracts several features that potentially differentiate it from other data points, forming a representation space. Then, this low-dimensional representation space is mapped to the label (output) space, which is essentially a labeled clustering process.

Therefore, to achieve good classification performance, we need to increase the distance between points belonging to different classes while reducing the distance between points of the same class. This concept is remarkably similar to the idea of contrastive learning, but in the presence of labeled data. From an information theory perspective, it means compressing the information content of data points of the same class as much as possible while ensuring distinguishability between different classes. From a linear algebra viewpoint, it involves within-class discrimination: features of the same class/cluster should be highly compressed in a low-dimensional linear subspace, and between-class discrimination: features of different classes/clusters should reside in highly unrelated linear subspaces. Following this line of thought, if we can calculate the number of bits required to encode the entire dataset, we can determine the size of the space needed to faithfully represent the dataset without loss. This, in turn, helps us establish appropriate distances for within-class and between-class comparisons. According to the paper given by MaYi from TPAMI'07, the following function can measure the least bits we should use to represent a dataset with a given loss rate ϵ:

$$L(X, \epsilon) \doteq (\frac{m + D}{2}) \log det(I + \frac{D}{m\epsilon^2} XX^T)$$

Following this function, Ma Yi derived following function: To maximize this target function, which also name "Maximial Coding Rate Reduction(MCR^2). We can straightforwardly understand this with the picture. As the figureE.2 shows, assume the dashed circle represents the size of the subspace needed for the entire dataset. The more blue balls there are, the better the feature compression result. This is because the total number of blue and green balls represents the spatial scale of the entire dataset $\mathbb{R}$, and the number of green balls represents the sum of the space required for each class $\mathbb{R}$. Therefore, the blue balls represent the redundant space scale $\Delta\mathbb{R}$. When MCR^2 was first introduced in a paper, it was combined with ResNet and essentially modified the loss function of ResNet. The specific details of its usage are not elaborated here. The experimental results were impressive, as it significantly improved the robustness of the model compared to the cross-entropy loss function (CE).



Professor Ma pointed out that although an information-theoretic perspective was employed, it was still a black-box model that required improvement. Therefore, the ReduNet network was proposed to achieve a fully white-box approach.

E.3. Extension of Rate Reduction algorithm: CRATE (Yu et al., 2023)(white-box transformer)

The overview of the Model is shown in the Figure E.3 Among them, the compression layer is called MSSA, which corresponds to the MHA layer in a Transformer. The specification layer is called ISTA, which corresponds to the FFN layer in a Transformer. Given an input, the first MSSA layer produces an output, which is then passed to the first ISTA layer, producing. This output is then passed to the second MSSA layer, producing, which is further passed to the second ISTA layer, producing. This iteration continues until the final layer. This white box transformer-like deep network architecture is an iterative unrolling optimization scheme to incrementally optimize the sparse rate reduction objective:

$$max_{f \in F} E_Z[\Delta R(Z; U_{[K]}) - \|Z\|_0], Z = F(X)$$

Which also can be written as:

$$\begin{aligned} &\arg\max_{f\in F} E_Z[\Delta R(Z; U_{[K]}) - \|Z\|_0] \\ =&\arg\min_{f\in F} E_Z[\underbrace{R^c(Z; U_{[K]})}_{\text{compression}} + \underbrace{\|Z\|_0 - R(Z)}_{\text{sparsification}}] \end{aligned} \tag{1}$$

Where $U_{[K]} = (U_1, ..., U_{[k]}),\ U_k \in R^{d\times p}$ are subspaces parameterizing the marginal distribution of tokens $(z_i)_{i=1}^N$ We can simply understand this process as following pictures:

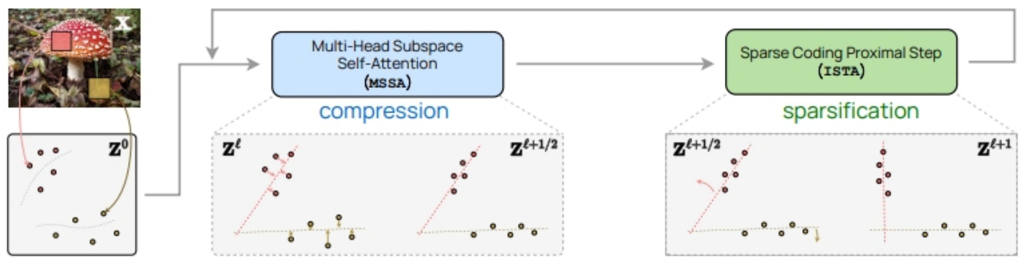


Figure 1: The 'main loop' of the CRATE white-box deep network design. After encoding input data X as a sequence of tokens Z^0, CRATE constructs a deep network that transforms the data to a canonical configuration of low-dimensional subspaces by successive ***compression*** against a local model for the distribution, generating $Z^{\ell+1/2}$, and ***sparsification*** against a global dictionary, generating $Z^{\ell+1}$. Repeatedly stacking these blocks and training the model parameters via backpropagation yields a powerful and interpretable representation of the data.

Then, the CRATE model simply interactively uses the sparsification layer and compression to optimize the objective function, the over view of the architecture is shown as follows.

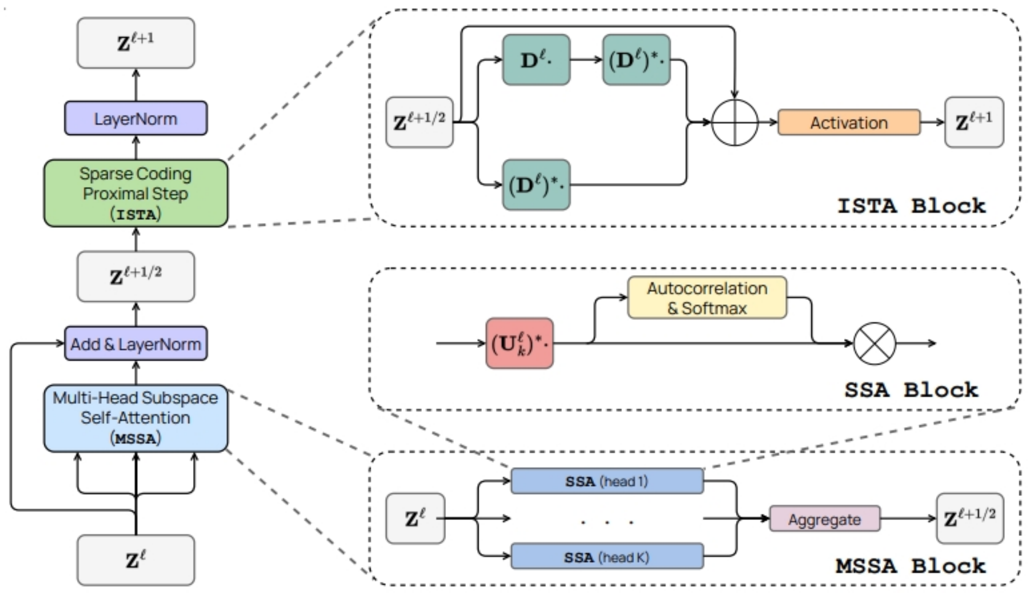


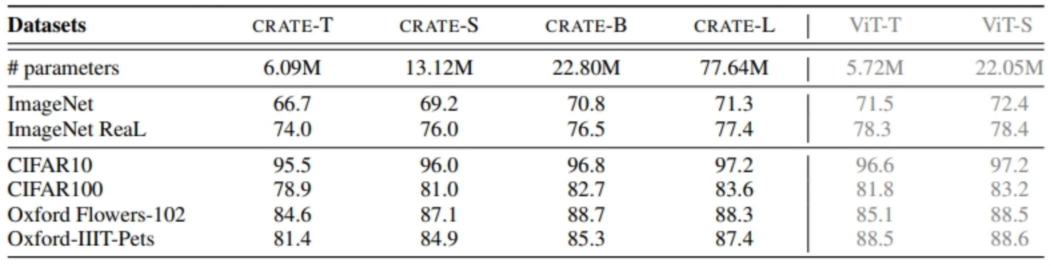
Figure 2: One layer of the CRATE architecture. The full architecture is simply a concatenation of such layers, with some initial tokenizer and final task-specific architecture (i.e., a classification head).

In the above picture the layer Z^l to the layer $Z^{l+\frac{1}{2}}$ is doing compression while the layer $Z^{l+\frac{1}{2}}$ to layer Z^{l+1} is used for compression. Combining multiple above layers will get the White-box transformer which converts tokens into a compact and sparse set of mutually independent subspaces, making classification easier.

In the training time, the CRATE apply SGD to learn($U^l_{[K]}, D^l)^L_{l=1}$ During backpropagation, $U^l_{[K]}$ is always the orthogonal basis of the subspace supporting GMM in the l-th layer, while D^l is always the dictionary used for sparsification in the l-th layer. Since each layer has these two parameters, each layer will learn different sets of parameters.

The experiment shows that CRATE performs similar accuracy with the ViT model with the similar parameters, the result shows as follows:

Table 1: Top 1 accuracy of CRATE on various datasets with different model scales when pre-trained on ImageNet. For ImageNet/ImageNetReaL, we directly evaluate the top-1 accuracy. For other datasets, we use models that are pre-trained on ImageNet as initialization and the evaluate the transfer learning performance via fine-tuning.



Datasets	CRATE-T	CRATE-S	CRATE-B	CRATE-L	ViT-T	ViT-S
# parameters	6.09M	13.12M	22.80M	77.64M	5.72M	22.05M
ImageNet	66.7	69.2	70.8	71.3	71.5	72.4
ImageNet ReaL	74.0	76.0	76.5	77.4	78.3	78.4
CIFAR10	95.5	96.0	96.8	97.2	96.6	97.2
CIFAR100	78.9	81.0	82.7	83.6	81.8	83.2
Oxford Flowers-102	84.6	87.1	88.7	88.3	85.1	88.5
Oxford-IIIT-Pets	81.4	84.9	85.3	87.4	88.5	88.6