Universidad ORT Uruguay
Facultad de Ingeniería

Exploring Attention Patterns and Neural Activations in Transformer Architectures for Sequence Classification in Context Free Grammars

Entregado como requisito para la obtención del título de Ingeniería en Sistemas

Matías Molinolo De Ferrari - 231323

Tutores: Dr. Sergio Yovine, Dr. Franz Mayr

2024

Declaración de Autoría

Yo, Matias Molinolo De Ferrari, declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el Proyecto Final de Ingeniería en Sistemas;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Matias Molinolo De Ferrari

dd-10-2024

Agradecimientos

{DEDICATORIA}

{AGRADECIMIENTOS}

Abstract

Cuerpo del Abstract.

Abstract Español

Cuerpo del Abstract.

Palabras clave

Inteligencia Artificial; Transformers; Redes neuronales; Lenguajes formales; Gramáticas formales; Autómatas a pila; Explicabilidad, Interpretabilidad; Expresividad

Key words

Artificial Intelligence; Transformers; Neural networks; Formal languages; Formal grammars; Pushdown automata; Explainability; Interpretability; Expressivity

Contents

1 Introduction

Large Language Models (LLMs) have been a topic of interest in the field of Computer Science for the past few years, and more recently, with the release of ChatGPT [**?**] by OpenAI, they have become a topic of interest for the general public too.

These models are based on an architecture called Transformer [**?**], a type of Artificial Neural Network (ANN) well suited to process sequences, such as text. These models have grown exponentially, as seen in 1.1, both in size and complexity, in the last years, and have shown to achieve state-of-the-art results in a wide variety of Natural Language Processing (NLP) and Natural Language Understanding (NLU) tasks.

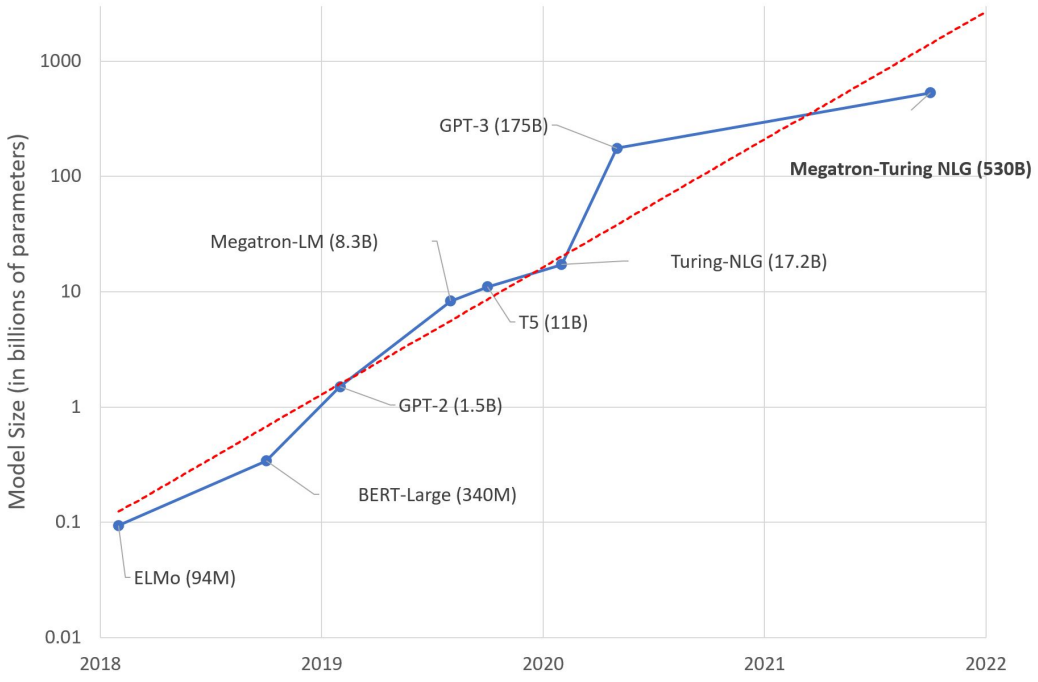

Figure 1.1: Model Sizes (2018–2021) [**?**]

However, despite their state-of-the-art performance, the inner workings of these models are not yet fully understood and these models are still considered opaque or *black-box* [**?**], which is a problem for their adoption in critical applications, such as healthcare or finance where decisions need to be explainable and interpretable.

Moreover, there is still a boundary to the capabilities of these models, regarding which problems can or cannot be solved by them and what can be learned and expressed by these models.

Strobl et. al [**?**] defines two lines of work done regarding this problem: *expressivity* and *trainability*. This work will focus on the latter, as we aim to determine whether through training and using a *white-box* approach, by looking at the attention mechanism and neural activations, whether Transformers are able to learn and classify sequences belonging to a formal language, more specifically, context-free grammars (CFGs) such as Dyck languages.

Chomsky's hierarchy [**?**] classifies CFGs as those languages that can be represented by a nondeterministic pushdown automaton, a class of automata that is more expressive than finite-state machines but less so than Turing machines [**?**]. Pérez, Barceló and Marinkovic propose that architectures based on self-attention, such as Transformers, are Turing complete [**?**], therefore, this leads us to believe that CFGs can be expressed by these neural language models.

Based on the aforementioned, we seek to discuss whether attention patterns and neural activations in these architectures can be used to explain the model's classifications, and if these explanations can be leveraged to improve the model's performance and interpretability.

Outline

Chapter 2 will introduce the concepts behind formal languages and context-free grammars, focusing especially on the Chomsky Hierarchy and Dyck-k languages.

Chapter 3 will introduce the Transformer architecture, with a focus towards the attention mechanism.

Chapter 4 will discuss related works and the state-of-the-art in the field of trainability of transformers, explainable AI and formal languages.

Chapter 5 focuses on the experimental setup, the dataset used, the model architecture, the training process and the obtained results.

Chapter 6 will discuss the results obtained and the implications of these results.

Chapter 7 will summarize the work done and propose future work.

2 On Formal Languages

Hopcroft, Motwani and Ullman define a language $\mathcal{L}$ as a set of strings chosen from Σ^*, where Σ is a particular alphabet. Σ^* denotes the *universal language*, which is the language formed by all possible sequences over an alphabet Σ. An alphabet is defined as a finite, nonempty set of symbols. [**?**]

Furthermore, they define a string or word as a finite sequence of symbols chosen from an alphabet Σ. Therefore, a word can also be seen as a concatenation of symbols that start with the *empty* or *identity* symbol, ϵ.

However, not every word formed by the alphabet necessarily belongs to a language, since for each language, a specific set of rules exist that define *membership* - whether a word belongs to the language or not.

Furthermore, it is important to note the existence of the *empty language*, $\varnothing$, that is, the language that does not contain any words, not even the empty one - $\epsilon \notin \varnothing$.

The only constraint on what can be a language is that, by definition, alphabets are finite, therefore, even though there might be an infinite amount of strings in a language, they all stem from a finite alphabet [**?**].

Chomsky defines a grammar as a "device that enumerates the sentences of a language" and proposes a set of restrictions that limit these grammars to different types of automata, in a way that the "languages that can be generated by grammars meeting a given restriction constitute a proper subset of those that can be generated by grammars meeting the preceding restriction" [**?**].

Furthermore, Chomsky proposes that these grammars, ordered by increasing restrictions, can be recognized by different automata - Turing machines, linear-bounded automata (LBA), pushdown automata (PDA) or Finite State Machines (FSM), as seen in 2.1.

For this work, we will focus on Type-2, or context-free grammars, which can be recognized by pushdown automata, which have a "natural, recursive notation" [**?**]. For example, palindromes are a context-free language, as the grammar that generates it is also context-free.

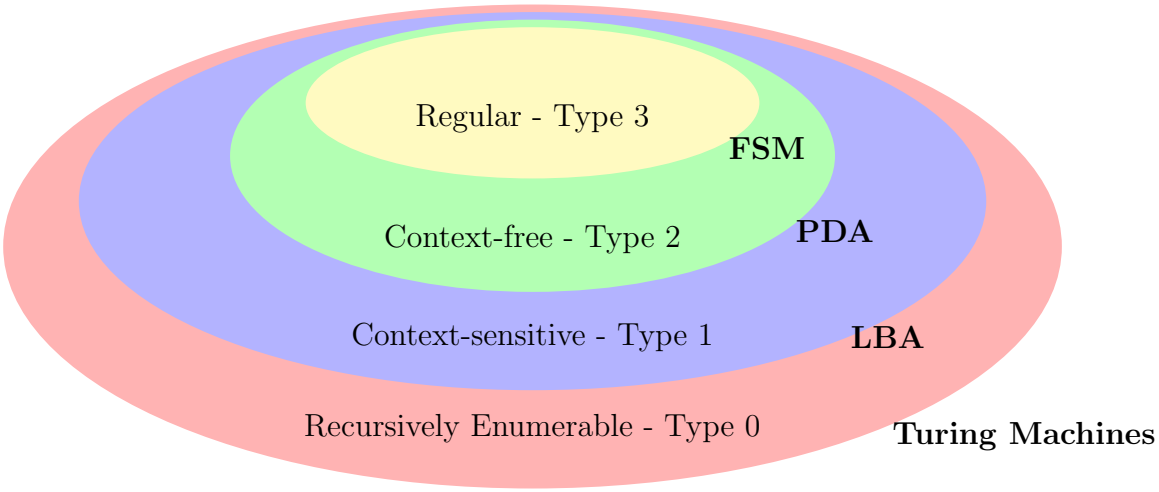

Figure 2.1: Chomsky Hierarchy

2.1 Context-free grammars

A context free grammar $\mathcal{G}$ has four main components [**?**, **?**]:

1. A finite set of *variables* $\mathbf{V}$ - these variables represent a language (a set of strings). We note $\mathbf{V}^*$ as the Kleene closure of the set of variables - in essence, the concatenation of zero or more repetitions of elements in the set.
2. A finite set of symbols $\mathbf{T}$, called *terminals*. Note that $\mathbf{T} \subset \mathbf{V}$. Once again, $\mathbf{T}^*$ is the Kleene closure of the set.
3. A *start symbol* $\mathbf{S} \in \mathbf{V} - \mathbf{T}$.
4. A finite set of *productions*, $\mathbf{P} \subset (\mathbf{V} - \mathbf{T}) \times \mathbf{V}^*$. These represent the recursive definition of the language. Each production consists of the following:
   (a) A variable, called the *head* that is partially defined by the production.
   (b) A production symbol $\rightarrow$
   (c) A string of zero or more terminals and variables, called the *body*.

Therefore, a grammar $\mathcal{G}$ can be expressed as a four-tuple $(\mathbf{V}, \mathbf{T}, \mathbf{P}, \mathbf{S})$.

Given 2 words, $u, v \in \mathbf{V}^*$, we say that $u \rightarrow v$ if there exists a derivation, or sequence of words in $\mathbf{V}^*$ such that $u_{i-1} \rightarrow u_i$ for $i = 1, \ldots, k$ and $u_0 = u$ and $v = u_k$. The existence of a derivation is noted by $u \xrightarrow{*} v$

Furthermore, we define the *language* $\mathcal{L}$ *generated by* $\mathcal{G}$ as the following set:

$$\mathcal{L}(\mathcal{G}) = \{w \in \mathbf{T}^* | \mathbf{S} \xrightarrow{*} w\}$$

Should X be a variable in $\mathcal{G}$, then:

$$\mathcal{L}_{\mathcal{G}}(X) = \{w \in \mathbf{T}^* | X \xrightarrow{*} w\}$$

From this, we derive that $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{\mathcal{G}}(S)$. If two grammars generate the same language, they are considered *equivalent* [**?**].

2.2 Dyck-k Languages

Dyck-k languages are a canonical family of context-free languages, composed of strings of balanced parentheses. A Dyck-k language is defined as a set of strings over the alphabet $\Sigma = \{a_1, \bar{a_1}, \dots a_n, \bar{a_n}\}$, where each string is a sequence of $2k$ parentheses, such that the parentheses are balanced. That is, for each string $w \in \Sigma^*$, the number of opening parentheses is equal to the number of closing parentheses, and for each prefix w' of w, the number of opening parentheses is greater than or equal to the number of closing parentheses.

The language of Dyck-k is denoted by D_k and is defined by the following grammar:

$$\begin{aligned}
\mathbf{V} &= \{S\} \\
\mathbf{T} &= \{a_1, \dots a_n\} \cup \{\bar{a_1}, \dots, \bar{a_n}\} \\
\mathbf{P} &= \{S \to a_i S \bar{a_i} S | \epsilon\} \text{ for } i = 1, \dots, n \\
\mathbf{S} &= \epsilon
\end{aligned}$$

In this case, $\mathbf{T}$ is called a *matched alphabet*, as each symbol a_i has a corresponding closing symbol $\bar{a}_i$.

We say that Dyck-k languages are canonical because they are the simplest form of context-free languages and can be used as a building block for all other context-free languages. This is known as the Chomsky-Schützenberger Theorem [**?**] and states that a language $\mathcal{L}$ over an alphabet Σ is context-free iff there exists:

- a matched alphabet $T \cup T', |T| = k$,
- a regular language $\mathcal{R}$ over $T \cup T'$,
- a homomorphism $h : (T \cup T')^* \to \Sigma^*$

such that $\mathcal{L} = h(R \cap D_k)$ for some integer k such that D_k is the Dyck language over k symbols.

It is useful to visualize a matched alphabet $T \cup T'$ as matched parentheses, with T being the set of opening parentheses and T' the set of closing ones.

We find Dyck-k languages interesting to study as they can showcase subject-verb agreement in common language (english, spanish, etc.) [**?**], therefore we can consider Dyck-k languages as a sort of building block for common language. We can see an example in 2.2:

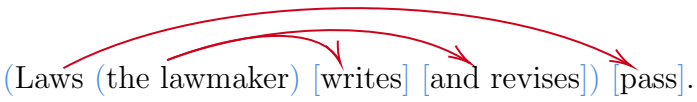

Figure 2.2: Subject-verb agreement [**?**]

We see that this subject-verb agreement can be expressed by the Dyck-2 word (( ) [ ] [ ]) [ ].

3 Transformer Architecture

Transformers [?] were proposed by Vaswani et al. in 2017. This novel architecture achieved state-of-the-art results in machine translation tasks, while dispensing with recurrent and convolutional architectures, which enabled parallelization and speedup of natural language processing tasks. The Transformer employs an encoder-decoder stack, each with multiple *blocks*, as can be seen in 3.1. The key mechanism behind this architecture is called *attention*, which we will discuss in detail in a later section.

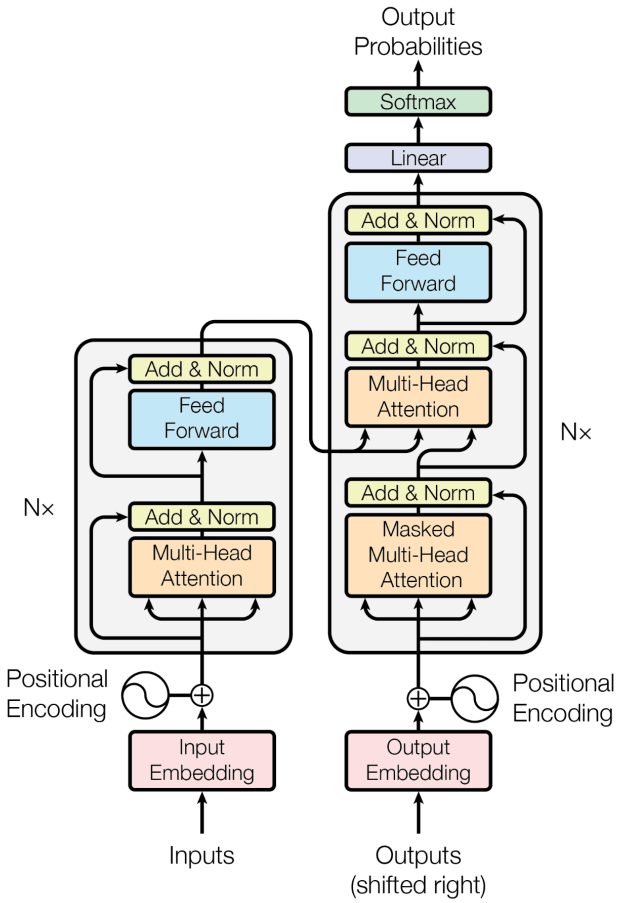

Figure 3.1: Transformer Architecture [?]

As we can see, the architecture lends itself to parallelization, since there are no recurrences or dependencies between inputs, which in turn helps speed up training

and inference, but also increases the model's complexity and reduces its interpretability, due to the existence of residual paths, also called *residual connections*, which are paths that add the input without processing to the output that stems from processing that same input, similar to doing $x = x + f(x)$, layer normalizations and attention mechanisms.

For our work, we will focus on Transformer encoder stacks (which we will call encoder-only Transformers hereon forwards) with multi-head attention, as these can be used for sequence classification rather than machine translation or sequence modeling.

These encoder-only Transformers, when used along a feed-forward layer and a softmax layer, can output n-class classification probabilities, which we will use to determine if a string belongs to a certain language, after being trained on a dataset composed of strings of balanced and unbalanced parentheses.

Albeit Transformers are known for their state-of-the-art performance on natural language processing tasks, they cannot process words as-is; these inputs need to be mapped to a vector space $\mathbb{R}^d$. To this effect, we will use Ströbl's definition of a *word embedding*, $\mathrm{WE} : \Sigma \to \mathbb{R}^d$ [**?**].

Figure 3.1 displays the full architecture of the original Transformer, with an encoder and decoder, each with its inner components, however, we will deal with a simplified version of this architecture, and we will proceed to explain each part that makes up our encoder-only Transformers.

3.1 Attention

The attention mechanism is the most important part of this architecture and will be the focus of our study. In short, this mechanism will let the model know the importance of a *token* with respect to all other tokens in the sequence.

This concept was introduced by Badhanau et. al. and is defined as an alignment model [**?**] that scores matches between inputs at a given position to outputs at another position. Even though this mechanism was applied to recurrent neural networks (RNNs), it is equivalent to the mechanism we will describe next.

According to Vaswani et al., attention functions can be described as mappings of queries and key-value pairs to an output, where queries, keys and values are all vectors belonging to a vector space $\mathbb{R}^n$, where n is called the *embedding* or *representation* dimension. As defined above, Ströbl defines these vectors as mappings that stem from applying a length preserving function $f : \Sigma^* \to (\mathbb{R}^d)^*$ to input strings [**?**]. This function consists of two components, the previously defined word embedding and a positional encoding, PE, such that:

$$f(w_0 \dots w_{n-1})_i = \mathrm{WE}(w_i) + \mathrm{PE}(i) \tag{3.1}$$

The most commonly used attention mechanism is called scaled dot-product attention or *soft(max)* attention, which is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

In practice, Q, K, V in equation 3.2 refer to batched queries, keys and values respectively, where individual queries, keys and values are packed into matrices and processed simultaneously, speeding up computation.

Furthermore, this mechanism can be split and parallelized, which is then known as *multi-head attention*, and represented by the following equation:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.3}$$

In this mechanism, $W_i^{\{Q,K,V\}}$ are learned parameter matrices. Splitting the attention mechanism into different *heads* allows for attending to information at different positions using different representations, without incurring in severe computational cost penalties, as the reduced dimensionality of each head allows for a computational cost similar to single-head attention with full dimensionality [**?**].

We can also define *hard* attention, which we define with the following equation:

$$\text{HardAttention}(Q, K, V) = \text{argmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.4}$$

We also note the difference between leftmost-hard and average-hard attention mechanisms, as the former looks for the first element with the maximum value, whereas the latter averages these and allows them to share weight equally [**?**].

Finally, we introduce the concept of *self-attention*, which is simply applying the mechanism to the input sequence, in essence, allowing us to see internal relations within the sequence [**?**].

We find this mechanism (*self-attention*) of interest for our work, as we believe attention will provide information regarding if a certain string of parentheses is balanced, and therefore, information about its membership to a D_k language.

3.2 Attention Masks

We define an attention mask as follows:

$$\begin{cases} 0 & \text{if condition is true} \\ -\infty & \text{otherwise} \end{cases} \tag{3.5}$$

We define 2 conditions of interest: $i \leq j$ and tok$[i] \neq$ `<PAD>` (pad token masking).

The first condition allows the mechanism to attend *only* to previous positions, such that the mechanism cannot get information on tokens it has not yet seen - we call this mechanism future or *causal* masking. The second condition allows for the attention mechanism to "see" the sequence as a whole, but considers only tokens that provide information, ignoring special `<PAD>` tokens, used to unify input sequence lengths.

We will analyze in further detail whether different attention masks have an effect on trainability in a later chapter.

4 Trainability of Transformers

Having already analyzed the Transformer architecture, we will now proceed with a discussion on whether these architectures are capable of being trained to recognise context-free grammars.

However, before we dive into this analysis on the *trainability* of Transformers, which will be backed by experimental data, we first must analyze whether these architectures are capable of learning context-free grammars or not - in essence, are they *expressive* enough to generate an internal model of a context-free grammar?

Bhattamistra et. al. [**?**] prove that these architectures are expressive enough to recognize the Shuffle-Dyck-k family of languages and notes the special case of $k = 1$, where Shuffle-Dyck-k is equal to Dyck-1 languages. More specifically, Ströbl et. al. [**?**] specifies that this Transformer is one with soft-attention (i.e.: softmax attention, as defined in **??**) with future masking, positional encoding only, no layer normalization and no residual connections.

In our experiments, we found that this Transformer may be expressive enough to recognize this family of languages, but is not trainable enough, however, a Transformer with soft-attention with future masking, positional encoding only, layer normalization and residual connections is trainable enough to recognize and generalize to both the training and test datasets, reaching 100% accuracy and a loss of 0 in both sets.

5 Annexes

5.1 Annex 1

5.2 Annex 2