ATTENTION IS A SMOOTHED CUBIC SPLINE

ZEHUA LAI, LEK-HENG LIM, AND YUCONG LIU

Abstract. We highlight a perhaps important but hitherto unobserved insight: The attention module in a transformer is a smoothed cubic spline. Viewed in this manner, this mysterious but critical component of a transformer becomes a natural development of an old notion deeply entrenched in classical approximation theory. More precisely, we show that with ReLU-activation, attention, masked attention, encoder–decoder attention are all cubic splines. As every component in a transformer is constructed out of compositions of various attention modules (= cubic splines) and feed forward neural networks (= linear splines), all its components — encoder, decoder, and encoder–decoder blocks; multilayered encoders and decoders; the transformer itself — are cubic or higher-order splines. If we assume the Pierce–Birkhoff conjecture, then the converse also holds, i.e., every spline is a ReLU-activated encoder. Since a spline is generally just C^2, one way to obtain a smoothed C^∞-version is by replacing ReLU with a smooth activation; and if this activation is chosen to be SoftMax, we recover the original transformer as proposed by Vaswani et al. This insight sheds light on the nature of the transformer by casting it entirely in terms of splines, one of the best known and thoroughly understood objects in applied mathematics.

The transformer [45] underlies many modern AI technologies in the current news cycle. Splines, on the other hand, are among the oldest tools in classical approximation theory, studied since the 1940s, and culminated in the 1980s [14] before taking on a new life in the form of wavelets (e.g., the celebrated Cohen–Daubechies–Feauveau wavelet [9] that underlies JPEG 2000 compression comes from a B-spline). Indeed, the word "spline" originally refers to the flexible wooden strip that serves as a bendable ruler for shipbuilders and draftsmen to draw smooth shapes since time immemorial; the Wright brothers had notably used such wooden splines to design their aircraft. It is therefore somewhat surprising that a notion so old is nearly one and the same as a notion so new — we will show that every ReLU-activated attention module $F : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ is a multivariate cubic spline, and, if we assume a conjecture of Garrett Birkhoff and Richard Pierce from 1956 [5], then conversely every multivariate spline $G : \mathbb{R}^m \to \mathbb{R}^n$ is a ReLU-activated encoder. The usual SoftMax-activated attention module is thus a simple and natural way to make a cubic spline, which is at most a C^2-function, into a smooth function — by replacing the nonsmooth ReLU with a smooth SoftMax.

Why did approximation theorists not discover the transformer then? We posit that it is due to a simple but fundamental difference in how they treat the decomposition of a complicated function into simpler ones. In approximation theory and harmonic analysis, one decomposes a complicated function F into a *sum* of simpler functions $f_1, \dots, f_r$,

$$F = f_1 + f_2 + \cdots + f_r; \tag{1}$$

in artificial intelligence, one decomposes F into a *composition* of simpler functions $F_1, \dots, F_r$,

$$F = F_1 \circ F_2 \circ \cdots \circ F_r. \tag{2}$$

This fundamental difference in modeling a function is a key to the success of modern AI models. Suppose $F : \mathbb{R}^n \to \mathbb{R}^n$. If we model F as a sum in (1), the number of parameters scales like nd^n:

$$F(x) = \sum_{i_1=1}^{d} \cdots \sum_{i_n=1}^{d} \sum_{j=1}^{n} a_{i_1 i_2 \cdots i_n j} \varphi_{i_1}(x_1)\varphi_{i_2}(x_2)\cdots\varphi_{i_n}(x_n) e_j;$$

whereas if we model F as a composition in (2), it scales like $dn^2 + (d-1)n$:

$$F(x) = A_d \sigma_{d-1} A_{d-1} \cdots \sigma_2 A_2 \sigma_1 A_1 x$$

with $A_i \in \mathbb{R}^{n\times n}$, σ_i parameterized by a vector in $\mathbb{R}^n$. Note that even if $d = 2$, the size $n2^n$ quickly becomes untenable. Evidently, these ball park estimates are made with some assumptions: The root cause of this notorious *curse of dimensionality* is that there are no good general ways to construct the basis function f_i in (1) except as a tensor product of low (usually one) dimensional basis functions, i.e., as $\varphi_{i_1} \otimes \varphi_{i_2} \otimes \cdots \otimes \varphi_{i_n} \otimes e_j$. The compositional model (2) allows one to circumvent this problem beautifully. Take the simplest case of a d-layer feed forward neural network, as we did above; then it is well known that d can be small [11, 25].

An important feature of (1) and (2) is that both work well with respect to derivative by virtue of linearity

$$DF = Df_1 + Df_2 + \cdots + Df_r$$

or chain rule

$$DF = DF_1 \circ DF_2 \circ \cdots \circ DF_r.$$

The former underlies techniques for solving various PDEs, whether analytically or numerically; the latter underlies the back-propagation algorithm for training various AI models (where the former also plays a role through various variants of the stochastic gradient descent algorithm). The bottom line is that the conventional way to view a cubic spline, as a sum of polynomials supported on disjoint polygonal regions or a sum of monomials, takes the form in (1). A ReLU-attention module is just the same cubic spline expressed in the form (2), and in this form there is a natural and straightforward way to turn it into a smooth function, namely, replace all nonsmooth F_i's with smooth substitutes — if we replace ReLU by SoftMax, we obtain the attention module as defined in [45]. This is a key insight of our article.

It is well-known [1] that a ReLU-activated feed forward neural network may be viewed as a *linear spline* expressed in the form of (2). When combined with our insight that a ReLU-activated attention module is a *cubic spline*, we deduce that every other intermediate components of the ReLU-transformer — encoder, decoder, encoder–decoder — are either cubic or higher-order spline, as they are constructed out of compositions and self-compositions of ReLU-activated feed forward neural networks and ReLU-activated attention modules.

A word of caution: We are not claiming that SoftMax would be a natural smooth replacement for ReLU. We will touch on this in Section 4. Indeed, according to recent work [48], this replacement may be wholly unnecessary — when it comes to transformers, ReLU would be an equally if not superior choice of activation compared with SoftMax.

0.1. Understanding transformers via splines.

Our main contribution is to explain a little-understood new technology using a well-understood old one. For the benefit of approximation theorists who may not be familiar with transformers or machine learning theorists who may not be familiar with splines, we will briefly elaborate.

The transformer has become the most impactful technology driving AI. It has revolutionized natural language processing, what it was originally designed for [45], but by this point there is no other area in AI, be it computer vision [19], robotics [50], autonomous vehicles [38], etc, that is left untouched by transformers. This phenomenal success is however empirical, the fundamental principles underlying the operation of transformers have remained elusive.

The attention module is evidently the most critical component within a transformer, a fact reflected in the title of the paper that launched the transformer revolution [45]. It is arguably the only new component — the remaining constituents of a transformer are ReLU-activated feed forward neural networks, which have been around for more than 60 years [40] and thoroughly investigated. Unsurprisingly, it is also the least understood. An attention module is still widely understood by way of "query, key, value" and a transformer as a flow chart, as in the article where the notion first appeared [45]. The main goal of our article is to understand the attention module in particular and the transformer in general, by tying them to one of the oldest and best-understood object in approximation theory.

Splines are a mature, well-understood technology that has been thoroughly studied and widely used [41, 42, 12, 13, 15, 8, 47, 44, 35, 14], one of our most effective and efficient methods for approximating known functions and interpolating unknown ones. They have numerous applications and we will mention just one: representing intricate shapes in computer graphics and computer-aided design. Readers reading a hard copy of this article are looking at fonts whose outlines are defined by splines [26]; those viewing it on screen are in addition using a device likely designed with splines [20]. Splines have ushered in a golden age of approximation theory, and were studied extensively c. 1960–1980, until wavelets supplanted them. One could not have asked for a better platform to understand a new technology like the attention module and transformer.

Nowhere is this clearer than our constructions in Section 3.3 to show that every spline is an encoder of a ReLU-transformer. These constructions reveal how each feature of the transformer — attention, heads, layers, feed forward neural networks — plays an essential role. We made every attempt to simplify and failed: Omit any of these features and we would not be able to recreate an arbitrary spline as an encoder. It were as if the inventors of transformer had designed these features not with any AI applications in mind but to construct splines as compositions of functions.

1. Mathematical description of the transformer

A transformer is typically presented in the literature as a flow chart [45, Figure 1]. We show a version in Figure 1.

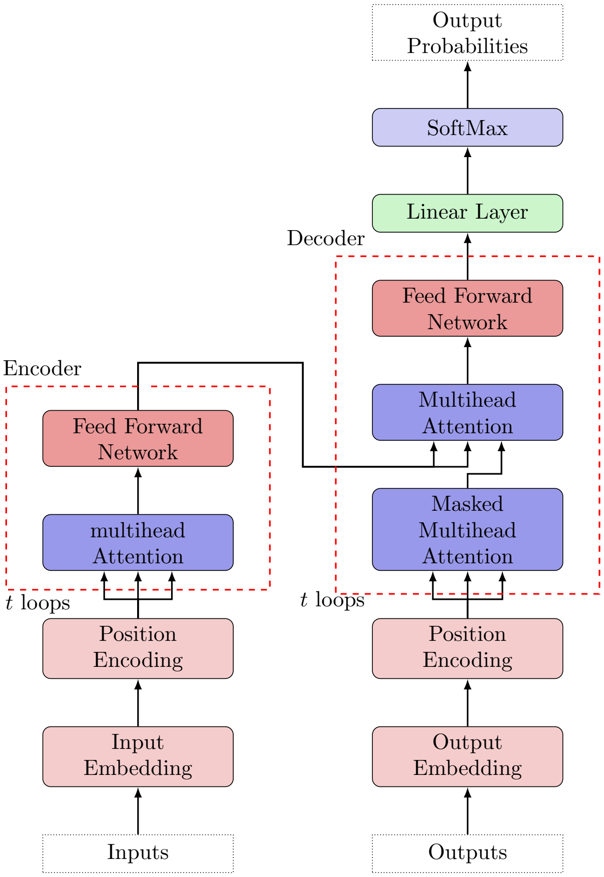


Figure 1. Transformer as flow chart.

Without a rigorous definition of the transformer, it will be difficult if not impossible to prove mathematical claims about it. We will nail down in mathematically precise terms the full inner workings of a transformer. While it is common to find descriptions that selectively present parts as well-defined maps and revert to words and pictures when it becomes less convenient, what sets us apart below is thoroughness — nothing will be swept under the rug. On occasions we had to look into the source codes of common implementations to unravel inconvenient details left ambiguous in the literature. This section is our small side contribution and a public service.

The heart of Figure 1 are the two parts enclosed in red dash lines, called encoder and decoder respectively. They are constructed out of feed forward neural networks, defined in Section 1.2, and attention modules, defined in Section 1.3, chained together via function compositions. The simplest version is the encoder in Section 1.4 and is what the uninitiated reader should keep in mind. We add the bells and whistles later: Section 1.5 defines the *masked* attention in the right-half of Figure 1, from which we obtain the decoder in Section 1.6. Section 1.7 explains the encoder–decoder structure — the left- and right-halves in Figure 1. Section 1.8 puts everything together to define the transformer. Section 1.10 discusses the one omission in Figure 1, the "add & norm" layers found in [45, Figure 1].

1.1. **Notations.** We write all vectors in $\mathbb{R}^n$ as column vectors, i.e., $\mathbb{R}^n \equiv \mathbb{R}^{n\times 1}$. Let $x_1, \dots, x_n \in \mathbb{R}$. When enclosed in parentheses $(x_1, \dots, x_n)$ denotes a *column* vector, i.e.,

$$(x_1, \dots, x_n) \coloneqq \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n.$$

When enclosed in brackets $[x_1, \dots, x_n] \in \mathbb{R}^{1\times n}$ is a *row* vector.

We will apply this convention more generally: For matrices $X_1, \dots, X_n \in \mathbb{R}^{m\times p}$, we write

$$(X_1, \dots, X_n) \coloneqq \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \in \mathbb{R}^{mn\times p}$$

and $[X_1, \dots, X_n] \in \mathbb{R}^{m\times np}$.

When we write $(f_1, \dots, f_h)$ for functions $f_i : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$, $i = 1, \dots, h$, it denotes the function

$$(f_1, \dots, f_h) : \mathbb{R}^{n\times p} \to \mathbb{R}^{mh\times p}, \quad X \mapsto \begin{bmatrix} f_1(X) \\ \vdots \\ f_h(X) \end{bmatrix}.$$

The function SoftMax : $\mathbb{R}^n \to \mathbb{R}^n$ takes a vector $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ and outputs a probability vector of the same dimension,

$$\operatorname{SoftMax}(x) \coloneqq \left(\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \dots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}} \right).$$

When SoftMax is applied to a matrix $X \in \mathbb{R}^{n\times p}$, it is applied columnwise to each of the p columns of X. So SoftMax : $\mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$.

Although we will write $\mathbb{R}$ throughout to avoid clutter, we will allow for $-\infty$ in the argument of our functions on occasion, which will be clearly indicated. Note that $\operatorname{SoftMax}(x)_i = 0$ if $x_i = -\infty$.

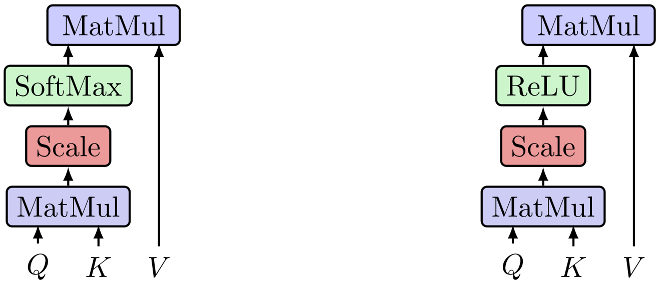
1.2. **Feed forward neural network.** The rectified linear unit ReLU : $\mathbb{R} \to \mathbb{R}$ is defined by $\operatorname{ReLU}(x) = \max(x, 0) =: x^+$ and extended coordinatewise to vectors in $\mathbb{R}^n$ or matrices in $\mathbb{R}^{n\times p}$. We also introduce the shorthand $x^- \coloneqq \operatorname{ReLU}(-x)$. Clearly, $X = X^+ - X^-$ for any $X \in \mathbb{R}^{n\times p}$.

An l-layer feed forward neural network is a map $\varphi : \mathbb{R}^n \to \mathbb{R}^{n_{l+1}}$ defined by a composition:

$$\varphi(x) = A_{l+1}\sigma_l A_l \cdots \sigma_2 A_2 \sigma_1 A_1 x + b_{l+1}$$

for any input $x \in \mathbb{R}^n$, weight matrix $A_i \in \mathbb{R}^{n_i\times n_{i-1}}$, with $n_0 = n$, $\sigma_i(x) \coloneqq \sigma(x + b_i)$, with $b_i \in \mathbb{R}^{n_i}$ the bias vector, and $\sigma : \mathbb{R} \to \mathbb{R}$ the activation function, applied coordinatewise. In this article, we set $\sigma = \text{ReLU}$ throughout. To avoid clutter we omit the $\circ$ for function composition within a feed forward neural network unless necessary for emphasis, i.e., we will usually write $A\sigma B$ instead of $A \circ \sigma \circ B$. When φ is applied to a matrix $X \in \mathbb{R}^{n\times p}$, it is always applied columnwise to each of the p columns of X. So $\varphi : \mathbb{R}^{n\times p} \to \mathbb{R}^{n_{l+1}\times p}$. We will also drop the "feed forward" henceforth since all neural networks that appear in our article are feed forward ones.

1.3. **Attention.** The *attention module* is known by a variety of other names, usually a combination of attention/self-attention module/mechanism, and usually represented as flow charts as in Figure 2.



(A) SoftMax attention (B) ReLU attention

FIGURE 2. Attention module as flow chart

Mathematically, it is a map $\alpha : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$,

$$\alpha(X) \coloneqq V(X)\,\text{SoftMax}\big(K(X)^{\mathsf{T}}Q(X)\big), \tag{3}$$

where $Q : \mathbb{R}^{n\times p} \to \mathbb{R}^{d\times p}$, $K : \mathbb{R}^{n\times p} \to \mathbb{R}^{d\times p}$, $V : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ are *linear layers*, i.e., given by affine maps

$$Q(X) = A_Q X + B_Q, \quad K(X) = A_K X + B_K, \quad V(X) = A_V X + B_V, \tag{4}$$

with weight matrices $A_Q, A_K \in \mathbb{R}^{d\times n}$, $A_V \in \mathbb{R}^{m\times n}$, and bias matrices $B_Q, B_K \in \mathbb{R}^{d\times p}$, $B_V \in \mathbb{R}^{m\times p}$. Here we have used the more general *affine* form of these linear layers as attention modules are implemented in practice,[1] as opposed to the *linear* form in [45] where the biases are set to zero. The SoftMax in (3) is applied columnwise and outputs a $p \times p$ matrix.

The map α implements the mechanism of taking a query and a set of key–value pairs to an output. Interpreted in this way, the input $X \in \mathbb{R}^{n\times p}$ is a data sequence of length p, with each data point $x_i \in \mathbb{R}^n$, $i = 1, \dots, p$. The columns of $Q(X)$ and $K(X)$ represent queries and keys respectively — note that these are vectors in $\mathbb{R}^d$ and d is generally much smaller than m or n. The columns of $V(X)$ represent values.

More generally, a multihead or h-headed attention module is a map $\alpha : \mathbb{R}^{n\times p} \to \mathbb{R}^{mh\times p}$ given by

$$\alpha(X) = (\alpha_1(X), \dots, \alpha_h(X)) \tag{5}$$

where $\alpha_i : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ are attention modules as in (3), $i = 1, \dots, h$. The reader is reminded of our convention in Section 1.1: parentheses denote column, which is why in our constructions we will often the phrase "stacking $\alpha_1, \dots, \alpha_h$ to obtain α" to mean (5).

[1] https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html#torch.nn.MultiheadAttention; the affine form is obtained by setting `add_bias_kv = True`.

1.4. **Encoder.** An *encoder block*, or more precisely a h-head encoder block, is a map $\varepsilon : \mathbb{R}^{n\times p} \to \mathbb{R}^{n_{l+1}\times p}$ obtained by composing the output of a h-head attention module $\alpha : \mathbb{R}^{n\times p} \to \mathbb{R}^{mh\times p}$, with an l-layer ReLU-neural network $\varphi : \mathbb{R}^{mh\times p} \to \mathbb{R}^{n_{l+1}\times p}$,

$$\varepsilon = \varphi \circ \alpha. \tag{6}$$

More generally, an *encoder* or t-layer encoder, $\varepsilon_t : \mathbb{R}^{n\times p} \to \mathbb{R}^{n_{t+1}\times p}$ is obtained by composing t encoder blocks, i.e.,

$$\varepsilon_t = \varphi_t \circ \alpha_t \circ \varphi_{t-1} \circ \alpha_{t-1} \circ \cdots \circ \varphi_1 \circ \alpha_1, \tag{7}$$

where $\varphi_i : \mathbb{R}^{m_i\times p} \to \mathbb{R}^{n_{i+1}\times p}$ are neural networks and $\alpha_i : \mathbb{R}^{n_i\times p} \to \mathbb{R}^{m_i\times p}$ are attention modules, $i = 1, \dots, t$, $n_1 = n$. In Figure 1, the encoder is the part enclosed within the red dash lines on the left. The structure in (7) appears to require alternate compositions of attention modules and neural networks but one may skip some or all of the φ_i's. The reason is that we may choose these φ_i's to be an identity map, which can be represented as a one-layer neural network as $x = \operatorname{ReLU}(x) - \operatorname{ReLU}(-x)$.

While we allow the neural networks appearing in (7) to have multiple hidden layers, the original proposed model in [45] requires that they be single-layer. We will show in Lemma 3.7 that these are in fact equivalent: Any encoder of the form (7) may be written as one where all φ_i's have only one hidden layer, but at the expense of a larger t.

1.5. **Masked attention.** In many applications of transformers, particularly large language models, the data is of a sequential nature. So the function f we want to learn or approximate is expected to be *autoregressive* [45], i.e., $f : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ takes the form

$$[x_1, \dots, x_p] \mapsto [f_1(x_1), f_2(x_1, x_2), \dots, f_p(x_1, \dots, x_p)]. \tag{8}$$

In other words $f_j : \mathbb{R}^{n\times j} \to \mathbb{R}^m$ depends only on the first j columns $x_1, \dots, x_j$, $j = 1, \dots, p$. In general f will be nonlinear, but when f is linear, then this simply means it is given by an upper triangular matrix. So an autoregressive function may be viewed as a nonlinear generalization of an upper triangular matrix.

To achieve this property in attention module, we define the function $\operatorname{mask} : \mathbb{R}^{p\times p} \to \mathbb{R}^{p\times p}$ by

$$\operatorname{mask}(X)_{ij} = \begin{cases} x_{ij} & \text{if } i \le j, \\ -\infty & \text{if } i > j. \end{cases}$$

A *masked attention* module is then given by

$$\beta(X) = V(X) \operatorname{SoftMax}\bigl(\operatorname{mask}(K(X)^{\mathsf{T}} Q(X))\bigr). \tag{9}$$

It is easy to check that a masked attention module is always autoregressive.

1.6. **Decoder.** A *decoder block* is the analogue of an encoder block where we have a masked attention in (6):

$$\delta = \varphi \circ \beta. \tag{10}$$

We may also replace any or all of the α_i's in (7) by masked versions β_i's. If we replace all, then the resulting map

$$\delta_t = \varphi_t \circ \beta_t \circ \varphi_{t-1} \circ \beta_{t-1} \circ \cdots \circ \varphi_1 \circ \beta_1, \tag{11}$$

is autoregressive but more generally we will just selectively replace some α_i's with β_i's. We call the resulting map a *decoder*. Note that the part enclosed within red dash lines in the right-half of Figure 1 is not quite a decoder as it takes a feed from the left-half; instead it is an *encoder–decoder*, as we will discuss next.

1.7. **Encoder–decoder attention.** The multihead attention in the right-half of Figure 1 accepts a feed from outside the red dash box. When used in this manner, it is called an *encoder–decoder attention module* [45], as it permits one to use queries from the decoder, but keys and values from the encoder. Mathematically, this is a map $\gamma : \mathbb{R}^{n\times p} \times \mathbb{R}^{r\times p} \to \mathbb{R}^{m\times p}$,

$$\gamma(X, Y) \coloneqq V(X)\,\mathrm{SoftMax}\bigl(K(X)^{\intercal}Q(Y)\bigr), \tag{12}$$

where Q, K, V are as in (4) but while K, V are functions of X, Q is now a function of Y. The independent matrix variables X and Y take values in $\mathbb{R}^{n\times p}$ and $\mathbb{R}^{r\times p}$ respectively. As a result we have to adjust the dimensions of the weight matrices slightly: $A_Q \in \mathbb{R}^{d\times r}$, $A_K \in \mathbb{R}^{d\times n}$, $A_V \in \mathbb{R}^{m\times n}$. The encoder–decoder attention is *partially autoregressive*, i.e., autoregressive in Y but not in X, taking the form

$$(X, [y_1, \dots, y_p]) \mapsto [f_1(X, y_1), f_2(X, y_2), \dots, f_p(X, y_1, \dots, y_p)].$$

1.8. **Transformer.** An *encoder–decoder block* $\tau : \mathbb{R}^{n\times p} \times \mathbb{R}^{r\times p} \to \mathbb{R}^{n_{l+1}\times p}$ is defined by a multihead masked attention module β, a multihead encoder–decoder attention module γ, and a neural network φ, via

$$\tau(X, Y) = \varphi\bigl(\gamma(X, \beta(Y))\bigr).$$

An $(s + t)$-layer *encoder–decoder* is then constructed from an s-layer encoder ε_s, and t encoder–decoder blocks given by $\beta_1, \gamma_1, \varphi_1, \dots, \beta_t, \gamma_t, \varphi_t$. We define τ_i recursively as

$$\tau_i(X, Y) = \varphi_i\Bigl(\gamma_i\bigl(\varepsilon_s(X), \beta_i(\tau_{i-1}(X, Y))\bigr)\Bigr)$$

for $i = 1, \dots, t$, $\tau_0(X, Y) = Y$. We call τ_t the encoder–decoder. For all mathematical intents and purposes, τ_t is the *transformer*. As we will see in Sections 1.10 and 1.11, the other components in Figure 1 or [45, Figure 1] are extraneous to the operation of a transformer.

We stress that the word "transformer" is sometimes used to refer to just the encoder or the decoder alone. We choose to make the distinction in our article but many do not. For example, Google's BERT [16], for Bidirectional Encoder Representations from Transformers, is an encoder whereas OpenAI's GPT [6], for Generative Pretrained Transformer, is a decoder.

1.9. **ReLU-transformer.** The definitions in Sections 1.2–1.8 are faithful mathematical transcriptions of components as described in Vaswani et al. original article [45]. In this section we take a small departure — replacing every occurrence of SoftMax with ReLU to obtain what is called a ReLU-transformer. This is not new either but proposed and studied in [3, 48].

We begin by defining ReLU-attention modules. They have the same structures as (3), (9), (12) except that SoftMax is replaced by ReLU, i.e.,

$$\begin{aligned} \alpha(X) &= V(X)\,\mathrm{ReLU}\bigl(K(X)^{\intercal}Q(X)\bigr),\\ \beta(X) &= V(X)\,\mathrm{ReLU}\bigl(\mathrm{mask}(K(X)^{\intercal}Q(X))\bigr),\\ \gamma(X, Y) &= V(X)\,\mathrm{ReLU}\bigl(K(X)^{\intercal}Q(Y)\bigr). \end{aligned} \tag{13}$$

An encoder, decoder, or encoder–decoder constructed out of such ReLU-attention modules will be called a ReLU-encoder, ReLU-decoder, or ReLU-encoder–decoder respectively. In particular, a ReLU-transformer is, for all mathematical intents and purposes, a ReLU-encoder–decoder.

These ReLU-activated variants are essentially "unsmoothed" versions of their smooth SoftMax-activated cousins in Sections 1.2–1.8. We may easily revert to the smooth versions by a simple smoothing process — replace all ReLU-activated attentions by the original SoftMax-activated ones (but the neural networks would remain ReLU-activated).

ReLU-transformers work naturally with our claims and proofs in Section 3. Nevertheless, even in practice ReLU-transformers can have desirable, possibly superior, features compared to the original SoftMax-transformers: investigations in [48] provided extensive empirical evidence that substituting SoftMax with ReLU causes no noticeable loss and occasionally even affords a slight gain in performance across both language and vision tasks; it is also easier to explain the in-context-learning capability of ReLU-transformers [3].

More generally, the use of alternative activations in a transformer is a common practice. There are various reasons to replace SoftMax, one of which is to avoid the considerable training cost associated with the use of SoftMax activation. In [30], SoftMax is replaced with a Gaussian kernel; in [27], only the normalization part of SoftMax is kept; in [22], it is shown that an activation does not need to map into the probability simplex. Linearized attentions are used in [39], and sparse attentions in [37]; these are intended primarily to accelerate the SoftMax operator but they have other features too.

1.10. **Layer normalization and residual connection.** Comparing our Figure 1 and [45, Figure 1], one might notice that we have omitted the "add & norm" layers.

The "add" step, also called residual connection [23], may be easily included in our analysis — all our results and proofs in Section 3 hold verbatim with the inclusion of residual connection. For an encoder block $\varepsilon : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$, a residual connection simply means adding the identity map $\iota : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$, $X \mapsto X$, i.e.,

$$\varepsilon + \iota : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}, \quad X \mapsto \varepsilon(X) + X,$$

and likewise for a decoder block $\delta : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$. For an encoder–decoder block $\tau : \mathbb{R}^{n\times p}\times\mathbb{R}^{r\times p} \to \mathbb{R}^{r\times p}$, a residual connection simply means adding the projection map $\pi : \mathbb{R}^{n\times p} \times \mathbb{R}^{r\times p} \to \mathbb{R}^{r\times p}$, $(X, Y) \mapsto Y$, i.e.,

$$\tau + \pi : \mathbb{R}^{n\times p} \times \mathbb{R}^{r\times p} \to \mathbb{R}^{r\times p}, \quad (X, Y) \mapsto \tau(X, Y) + Y.$$

As will be clear from the proofs in Section 3, all results therein hold with or without residual connection.

The "norm" step, also called layer normalization [2] refers to statistical standardization, i.e., mean centering and scaling by standard deviation of each column vector in X. This is an ubiquitous process routinely performed in just about any procedure involving any data for practical reasons. But this innocuous process introduces additional nonlinearity that does not fit in our framework.

We do not consider either of these critical to the workings of a transformer. They are by no means unique and may be easily replaced with other data standardization process, as shown in [22].

1.11. **Miscellany.** The "input/output embedding" and "position embedding" in Figure 1 convert sentences or images (or whatever real-world entity the transformer is used for) to an input in $\mathbb{R}^{n\times p}$; the "linear layer" and "SoftMax" in the right half assign probability values to the output. These are just auxiliary components necessary in any situation involving human-generated input or requiring human-interpretable output. They are common to all practical AI models and we do not regard them as part of the transformer architecture.

2. Splines

This section covers the salient aspects of splines relevant for us. We write $\mathbb{R}[x_1, \dots, x_n]$ for the ring of polynomials with real coefficients in variables $(x_1, \dots, x_n) =: x$ and $\mathbb{R}[x_{11}, \dots, x_{np}]$ for that in $(x_{ij})_{i,j=1}^{n,p} =: X$.

Splines have a rich history and a vast literature in applied and computational mathematics, this being precisely the reason we chose them as our platform to understand a new technology like the transformer. Mathematical *splines*, as opposed to the mechanical ones used by draftsmen and shipbuilders, were first named in [42]. A one-line summary of its early history, with many regretful omissions, is that univariate splines were first proposed in [41], multivariate splines in [4], B-Splines in [10], and box splines in [15].

An important departure of our discussion of splines in this article is that we will not concern ourselves with differentiability, avoiding the usual efforts to ensure that a piecewise-defined function is C^r at points where the different pieces meet. The reason is simple: our results in the next section will show that every continuous spline is a ReLU-transformer (and vice versa) and when presented as such, there is a straightforward and natural way to smooth a spline to any desired degree-of-smoothness r, namely, by replacing ReLU with a C^r-activation. So there is no need for us to even introduce the notions of knots, tangential continuity, curvature continuity, etc. Indeed, viewed in this manner, the transformer with its SoftMax activation is the first example of a "C^∞-spline" — an impossible object in classical constructions of splines as the degree-of-smoothness of a spline can never exceed the degree of its polynomial pieces.

2.1. **Scalar-valued splines.** In its simplest form a spline is a piecewise-polynomial real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ defined over a partition of its domain $\mathbb{R}^n$. The classical and most basic partition is a triangulation, i.e., a subdivision into n-dimensional simplices whose union is $\mathbb{R}^n$ and intersecting only along faces; more generally one may also use convex polytopes in place of simplices [13, 8, 35]. We will need a slightly more sophisticated partition called a semialgebraic partition [18, 17, 43]. For any $b \in \mathbb{N}$, let

$$\Theta_b \coloneqq \bigl\{\theta : \{1, \dots, b\} \to \{1, 0, -1\}\bigr\}, \tag{14}$$

a finite set of size 3^b. Note that this is really just the set of ternary numerals with b (ternary) bits.

Definition 2.1 (Partition)**.** Any $\pi_1, \dots, \pi_b \in \mathbb{R}[x_1, \dots, x_n]$ induces a sign partition of $\mathbb{R}^n$ via

$$\Pi_\theta \coloneqq \{x \in \mathbb{R}^n : \operatorname{sgn}(\pi_i(x)) = \theta(i),\ i = 1, \dots, b\}.$$

Then $\{\Pi_\theta : \theta \in \Theta_b\}$ is a partition of $\mathbb{R}^n$, the *semialgebraic partition* induced by $\pi_1, \dots, \pi_b$.

Note that the domain of θ in (14) merely serves as a placeholder for any b-element set and does not need to be $\{1, \dots, b\}$. Indeed we will usually write $\theta : \{\pi_1, \dots, \pi_b\} \to \{1, 0, -1\}$ to emphasize that it is an index for the partition induced by $\pi_1, \dots, \pi_b$. Any triangulation or partition into polytopes can be obtained by choosing appropriate linear polynomials $\pi_1, \dots, \pi_b$ so Definition 2.1 generalizes the basic one that requires partition to be piecewise linear.

Definition 2.2 (Spline)**.** Let $\{\Pi_\theta : \theta \in \Theta_b\}$ be the semialgebraic partition induced by $\pi_1, \dots, \pi_b \in \mathbb{R}[x_1, \dots, x_n]$. A continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is a *polynomial spline* of degree k if for each $i = 1, \dots, b$,

(i) π_i has degree not more than k;

(ii) if $\Pi_\theta \neq \varnothing$, then f restricts to a polynomial of degree not more than k on Π_θ, i.e., $f(x) = \xi_\theta(x)$ for all $x \in \Pi_\theta$, for some $\xi_\theta \in \mathbb{R}[x_1, \dots, x_n]$ of degree not more than k.

Henceforth, "spline" will mean "polynomial spline," "degree-k" will mean "degree not more than k," and "partition" will mean "semialgebraic partition." The small cases $k = 1, 2, 3, 5$ are customarily called linear, quadratic, cubic, and quintic splines respectively. The standard notation for the set of all r-times differentiable degree-k splines with partition induced by $\pi_1, \dots, \pi_b$ is $S_k^r(\pi_1, \dots, \pi_b)$ but since we will only need the case $r = 0$ and splines as defined in Definition 2.2 are always continuous, we may drop the superscript r.

Observe that $S_k(\pi_1, \dots, \pi_b)$ is a finite-dimensional real vector space. So it is straightforward to extend Definition 2.2 to $\mathbb{V}$-valued splines $f : \mathbb{R}^n \to \mathbb{V}$ for any finite-dimensional real vector space $\mathbb{V}$ using tensor product, namely, they are simply elements of $S_k(\pi_1, \dots, \pi_b) \otimes \mathbb{V}$ [29, Example 4.30]. For the benefit of readers unfamiliar with tensor product constructions, we go over this below in a concrete manner for $\mathbb{V} = \mathbb{R}^n$ and $\mathbb{R}^{n \times p}$.

2.2. **Vector-valued splines.** A vector-valued degree-k spline $f : \mathbb{R}^n \to \mathbb{R}^m$ is given by

$$f(x) = \sum_{i=1}^{m} f_i(x) e_i \quad \text{for all } x \in \mathbb{R}^n,$$

where $f_1, \dots, f_m \in S_k(\pi_1, \dots, \pi_b)$ and $e_1, \dots, e_m \in \mathbb{R}^m$ are the standard basis vectors. This is equivalent to requiring f be a degree-k spline coordinatewise, i.e., $f = (f_1, \dots, f_m)$ where $f_1, \dots, f_m \in S_k(\pi_1, \dots, \pi_b)$.

Traditionally, vector-valued splines are the most important class of splines for practical applications. Special cases include spline curves ($n = 1$, $m = 2$ or 3) and spline surfaces ($n = 2$, $m = 2$ or 3), used to parameterize curves and surfaces that pass near a collection of given data points. These are of fundamental importance in computer graphics and computer-aided design [20, 44].

2.3. **Matrix-valued splines.** In this case we are interested in splines that are not just matrix-valued but also matrix-variate. One nice feature with our treatment of splines in Section 2.1 is that we can define matrix-variate splines over $\mathbb{R}^{n \times p}$ by simply replacing all occurrences of $\mathbb{R}[x_1, \dots, x_n]$ with $\mathbb{R}[x_{11}, \dots, x_{np}]$. A matrix-valued degree-k spline $f : \mathbb{R}^{n \times p} \to \mathbb{R}^{m \times p}$ is then given by

$$f(X) = \sum_{i=1}^{m} \sum_{j=1}^{p} f_{ij}(X) E_{ij} \quad \text{for all } X \in \mathbb{R}^{n \times p}, \tag{15}$$

where $f_{ij} \in S_k(\pi_1, \dots, \pi_b)$ and $E_{ij} \in \mathbb{R}^{m \times p}$, $i = 1, \dots, m$, $j = 1, \dots, p$. Here E_{ij} is the standard basis matrix with one in (i, j)th entry and zeros everywhere else. Again, an alternative but equivalent way to define them would be in a coordinatewise fashion, i.e., $f = (f_{ij})_{i,j=1}^{m,p}$ where $f_{ij} \in S_k(\pi_1, \dots, \pi_b)$, $i = 1, \dots, m$, $j = 1, \dots, p$. Note that $p = 1$ reduces to the case in Section 2.2.

2.4. **Pierce–Birkhoff conjecture.** Garrett Birkhoff, likely the person first to realize the importance of splines in applications though his consulting work [49], also posed one of the last remaining open problems about splines [5].

Conjecture 2.3 (Pierce–Birkhoff). *For every spline $f : \mathbb{R}^n \to \mathbb{R}$, there exists a finite set of polynomials $\xi_{ij} \in \mathbb{R}[x_1, \dots, x_n]$, $i = 1, \dots, m$, $j = 1, \dots, p$ such that*

$$f = \max_{i=1,\dots,m} \min_{j=1,\dots,p} \xi_{ij}. \tag{16}$$

This conjecture is known to be true for $n = 1$ and 2 but is open for all higher dimensions [33]. Our results in Section 3.3 will be established on the assumption that the Pierce–Birkhoff conjecture holds true for all n, given that there is significant evidence [31, 46, 34] for its validity.

The kind of functions on the right of (16) we will call *max-definable functions* in the variables $x_1, \dots, x_n$. These are functions $f : \mathbb{R}^n \to \mathbb{R}$ generated by $1, x_1, \dots, x_n$ under three binary operations: addition $(x, y) \mapsto x + y$, multiplication $(x, y) \mapsto x \cdot y$, maximization $(x, y) \mapsto \max(x, y)$; and scalar multiplication $x \mapsto \lambda x$ by $\lambda \in \mathbb{R}$. Note that minimization comes for free as $\min(x, y) := -\max(-x, -y)$. Using the identity $xy^+ = \max\bigl(\min(xy, x^2y + y), \min(0, -x^2y - y)\bigr)$, any max-definable functions can be reduced to the form $\max_{i=1,\dots,m} \min_{j=1,\dots,p} \xi_{ij}$ with $\xi_{ij} \in \mathbb{R}[x_1, \dots, x_n]$ [24]. The notion may be easily extended to matrix-variate, matrix-valued functions $f : \mathbb{R}^{n \times p} \to \mathbb{R}^{m \times p}$ coordinatewise, i.e., by requiring that each $f_{ij} : \mathbb{R}^{n \times p} \to \mathbb{R}$ be a max-definable function in the variables $x_{11}, x_{12}, \dots, x_{np}$.

Clearly, the set of max-definable functions is contained within the set of splines. Pierce–Birkhoff conjecture states that the two sets are equal. Both are examples of an "f-ring" as defined in [5], now christened "Pierce–Birkhoff ring" after the two authors. If we drop multiplication from the list of binary operations generating the max-definable functions, the resulting algebraic object is the renown max-plus algebra or tropical semiring [32].

3. Equivalence of splines and transformers

We will show that every component of the transformer defined in Section 1 is a spline — neural network, attention module, masked attention module, encoder block, decoder block, encoder, decoder, encoder–decoder — so long as they are ReLU-activated. More importantly, if Conjecture 2.3 is true, then the converse also holds in the sense that every spline is an encoder. The equivalence between ReLU-activated feed-forward neural networks and linear splines is well-known [1]. The other equivalences will be established below. Henceforth we will assume ReLU-activation throughout this section and will not specify this unless necessary for emphasis.

3.1. **Transformers are splines.** We will first remind readers of the main result in [1] establishing equivalence between neural networks and linear splines.

Theorem 3.1 (Arora–Basu–Mianjy–Mukherjee)**.** *Every neural network $\varphi : \mathbb{R}^n \to \mathbb{R}$ is a linear spline, and every linear spline $\ell : \mathbb{R}^n \to \mathbb{R}$ can be represented by a neural network with at most $\lceil \log_2(n+1) \rceil + 1$ depth.*

Compositions of spline functions are by-and-large uncommon in the literature for reasons mentioned in the beginning — one usually combines splines by taking sums or linear combinations. Matrix-valued splines also appear to be somewhat of a rarity in the literature. Consequently we are unable to find a reference for what ought to be a fairly standard result about degrees under composition and matrix multiplication, which we state and prove below.

Lemma 3.2. (i) *Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be a spline of degree k and $f : \mathbb{R}^m \to \mathbb{R}^p$ a spline of degree k'. Then $f \circ g$ is a spline of degree kk'.*
(ii) *Let $f : \mathbb{R}^{r\times s} \to \mathbb{R}^{m\times n}$ and $g : \mathbb{R}^{r\times s} \to \mathbb{R}^{n\times p}$ be splines of degrees k and k'. Then $fg : \mathbb{R}^{r\times s} \to \mathbb{R}^{m\times p}$, $X \mapsto f(X)g(X)$, is a spline of degree $k + k'$.*

Proof. We first assume that $p = 1$, i.e., $f : \mathbb{R}^m \to \mathbb{R}$ is a spline of degree k'. For a degree-k spline $g = (g_1, \dots, g_m) : \mathbb{R}^n \to \mathbb{R}^m$, we claim that the composition $f \circ g$ is a spline of degree at most kk'.

A partition induced by any $\pi_1, \dots, \pi_b$ can be *refined* to $\pi_1, \dots, \pi_b, \pi_{b+1}, \dots, \pi_{b+c}$ by adding finitely many polynomials. Any spline in $S_k(\pi_1, \dots, \pi_b)$ is also a spline in $S_k(\pi_1, \dots, \pi_{b+c})$. By passing through such refinements, we may assume that $g_1, \dots, g_m$ are defined over a common partition. So let $g_1, \dots, g_m \in S_k(\pi_1, \dots, \pi_b)$ with

$$g_i(x) = \xi_{i,\theta}(x) \quad \text{for } x \in \Pi_\theta, \quad \theta \in \Theta_b$$

where $\Theta_b = \bigl\{\theta : \{\pi_1, \dots, \pi_b\} \to \{-1, 0, 1\}\bigr\}$. Let $f \in S_k(\rho_1, \dots, \rho_c)$ with

$$f(x) = \zeta_\phi(x) \quad \text{for } x \in \Pi_\phi, \quad \phi \in \Phi_c$$

where $\Phi_c = \bigl\{\phi : \{\rho_1, \dots, \rho_c\} \to \{-1, 0, 1\}\bigr\}$. Let $L \coloneqq \{\pi_1, \dots, \pi_b\}$ and

$$M \coloneqq L \cup \{\rho_j \circ (\xi_{1,\theta}, \dots, \xi_{m,\theta}) : j = 1, \dots, c,\ \theta \in \Theta_b\}.$$

Any $\phi : M \to \{1, 0, -1\}$ can be restricted to L, giving $\phi|_L : L \to \{1, 0, -1\}$. Let

$$H \coloneqq \{\rho_j \circ (\xi_{1,\phi|_L}, \dots, \xi_{m,\phi|_L}) : j = 1, \dots, c\} \subseteq L.$$

Then ϕ can also be restricted to H, giving $\phi|_H : H \to \{1, 0, -1\}$. For any nonempty Π_ϕ, we have

$$f \circ g(x) = \zeta_{\phi|_H} \circ (\xi_{1,\phi|_L}, \dots, \xi_{m,\phi|_L})(x)$$

for $x \in \Pi_\phi$ where $\phi \in \Phi_c$. So $f \circ g \in S_{kk'}(M)$. This shows (i) for $p = 1$. For general p, we may again assume, by passing through a refinement if necessary, that $f_1, \dots, f_p$ share a common partition, we then apply the same argument coordinatewise.

We then deduce (ii) from (i), by composing the spline (f, g) with the polynomial (and therefore spline) function $(X, Y) \to XY$. □

With the ground work laid in Section 1, i.e., having the components of a transformer rigorously defined, it becomes relatively straightforward to show that these components are all splines.

Theorem 3.3 (Components of a transformer as splines)**.**

(i) *An attention module is a cubic spline.*

(ii) *A masked attention module is a cubic spline.*
(iii) *An encoder–decoder attention module is a cubic spline.*
(iv) *An encoder block is a cubic spline.*
(v) *A decoder block is a cubic spline.*
(vi) *An encoder–decoder block is a quintic spline.*
(vii) *A t-layer encoder is a spline of degree 3^t.*
(viii) *A t-layer decoder is a spline of degree 3^t.*
(ix) *An encoder–decoder with s-layer of encoder blocks and t-layer of encoder–decoder blocks is a spline of degree $3^{t+s} + 3^t - 3^s$.*

Proof. Let $f, g : \mathbb{R}^n \to \mathbb{R}$ be splines of degree k. Since $x + y$, $\max(x, y)$ are linear spline, it follows from Lemma 3.2(i) that $f + g$ and $\max(f, g)$ are splines of degree k. In the attention module, $K(X), Q(X), V(X)$ are linear splines, it follows from Lemma 3.2(ii) that $K(X)^\mathsf{T}Q(X)$ is a quadratic spline. Hence $\operatorname{ReLU}(K(X)^\mathsf{T}Q(X)) = \max(K(X)^\mathsf{T}Q(X), 0)$ is also a quadratic splines and $\alpha(X) = V(X)\operatorname{ReLU}(K(X)^\mathsf{T}Q(X))$ is a cubic spline. Similarly, the masked attention $\beta(X)$ and encoder–decoder attention $\gamma(X, Y)$ in (13) are also cubic splines. Note that the encoder–decoder attention is a quadratic spline with respect to the first variable X, and a linear spline with respect to the second variable Y; but overall it is a cubic spline with respect to (X, Y).

A neural network is a linear spline by Theorem 3.1. So the encoder block in (6) and decoder block in (10) remain cubic splines by Lemma 3.2(i). The encoder–decoder block $\tau(X, Y) = \varphi(\gamma(X, \beta(Y)))$ is quadratic in X and cubic in Y, and thus quintic in (X, Y). Since a t-layer encoder or decoder is a composition of (masked) attention modules and neural networks, it is a spline of degree 3^t. For an encoder–decoder with s layers of encoder blocks and t layers of encoder–decoder blocks, induction on t gives $2 \times 3^s + 3 \times (3^{t+s-1} + 3^{t-1} - 3^s) = 3^{t+s} + 3^t - 3^s$ as its degree. □

The splines in (ii), (iii), (v), (viii) are autoregressive and those in (vi) and (ix) partially autoregressive. The term "autoregressive spline" does appear in the literature but it is used in a sense entirely unrelated to (8). We will have more to say about this in Corollary 3.10.

3.2. **Veronese map.** The degree-k Veronese embedding v_k is a well-known map in algebraic geometry [21, pp. 23–25] and polynomial optimization [28, pp. 16–17]. Informally it is the map that takes variables $x_1, \dots, x_n$ to the monomials of degree not more than k in $x_1, \dots, x_n$. This defines an injective smooth function

$$v_k : \mathbb{R}^n \to \mathbb{R}^{\binom{n+k}{k}}, \quad (x_1, \dots, x_n) \mapsto (1, x_1, \dots, x_n, x_1^2, x_1x_2, \dots, x_n^k). \tag{17}$$

The value $\binom{n+k}{k}$ gives the number of monomials in n variables of degree not more than k. Two simple examples: $v_k : \mathbb{R} \to \mathbb{R}^k$, $v_k(x) = (1, x, x^2, \dots, x^k)$; $v_2 : \mathbb{R}^2 \to \mathbb{R}^6$, $v_2(x, y) = (1, x, y, x^2, xy, y^2)$.

In algebraic geometry [21, pp. 23–25] the Veronese map is usually defined over projective spaces whereas in polynomial optimization [28, pp. 16–17] it is usually defined over affine spaces as in (17). Nevertheless this is a trivial difference as the former is just a homogenized version of the latter.

As is standard in algebraic geometry and polynomial optimization alike, we leave out the domain dependence from the notation v_k to avoid clutter, e.g., the quadratic Veronese map $v_2 : \mathbb{R}^2 \to \mathbb{R}^6$ and $v_2 : \mathbb{R}^6 \to \mathbb{R}^{28}$ are both denoted by v_2. This flexibility allows us to compose Veronese maps and speak of $v_k \circ v_{k'}$ for any $k, k' \in \mathbb{N}$. For example we may write $v_2 \circ v_2 : \mathbb{R}^2 \to \mathbb{R}^{28}$, using the same notation v_2 for two different maps.

The Veronese map is also defined over matrix spaces: When applied to matrices, the Veronese map simply treats the coordinates of an $n \times p$ matrix as np variables. So $v_k : \mathbb{R}^{n \times p} \to \mathbb{R}^{\binom{np+k}{k}}$ is given by

$$v_k(X) = (1, x_{11}, x_{12}, \dots, x_{np}, x_{11}^2, x_{11}x_{12}, \dots, x_{np}^k).$$

For example $v_2 : \mathbb{R}^{2\times 2} \to \mathbb{R}^{15}$ evaluated on $\left[\begin{smallmatrix} x & y \\ z & w \end{smallmatrix}\right]$ gives

$$(1, x, y, z, w, x^2, xy, xz, xw, y^2, yz, yw, z^2, zw, w^2).$$

An important observation for us is the following.

Lemma 3.4. *Let $k, k' \in \mathbb{N}$. Then every coordinate of $v_{kk'}(X)$ occurs in $v_k(v_{k'}(X))$.*

Proof. This is a consequence of the observation that any monomial of degree not more than kk' can be written as a product of k monomials, each with degree not more than k'. □

Another result that we will need is the following equivalent formulation of Pierce–Birkhoff conjecture in terms of Veronese map.

Lemma 3.5. *The Pierce–Birkhoff conjecture holds if and only if for any spline $f : \mathbb{R}^n \to \mathbb{R}$, there exist $k \in \mathbb{N}$ and a linear spline $\ell : \mathbb{R}^{\binom{n+k}{k}} \to \mathbb{R}$ such that $f = \ell \circ v_k$.*

Proof. Firstly note that Pierce–Birkhoff conjecture holds for $k = 1$: Any linear spline ℓ can be represented in the form $\min_i \max_j \xi_{ij}$ where ξ_{ij} are linear polynomials [36]. Conversely, if ℓ can be represented in the form $\min_i \max_j \xi_{ij}$, then it is clearly a linear spline.

Assuming that Pierce–Birkhoff conjecture holds in general, then any polynomial spline f can be written as $\min_i \max_j \xi_{ij}$, which is a linear spline over monomials of ξ_{ij}, i.e., $f = \ell \circ v_k$ for some linear spline ℓ. Conversely, if every polynomial spline f can be written as $\ell \circ v_k$ for some linear spline ℓ, then since ℓ can always be written as $\ell = \min_i \max_j \xi_{ij}$, we have $f = \min_i \max_j \xi_{ij} \circ v_k$ for some linear polynomials ξ_{ij}'s. Thus we recover the statement of Pierce–Birkhoff conjecture. □

Observe that Lemma 3.5 applies verbatim to matrix-variate splines $f : \mathbb{R}^{n \times p} \to \mathbb{R}$, except that n would have to be replaced by np throughout and we have

$$v_k : \mathbb{R}^{n \times p} \to \mathbb{R}^{\binom{np+k}{k}}, \quad \ell : \mathbb{R}^{\binom{np+k}{k}} \to \mathbb{R}. \tag{18}$$

3.3. **Splines are transformers.** We will show that any matrix-valued spline $f : \mathbb{R}^{n \times p} \to \mathbb{R}^{r \times p}$ is an encoder. First we will prove two technical results. We will use $i, \hat{\imath}, \bar{\imath}, j, \hat{\jmath}, \bar{\jmath}$ to distinguish between indices. We remind the reader that $x^+ := \operatorname{ReLU}(x)$.

Lemma 3.6 (Quadratic Veronese as encoders)**.** *Let $v_2 : \mathbb{R}^{n \times p} \to \mathbb{R}^{(np+2)(np+1)/2}$ be the quadratic Veronese map. There exists a two-layer encoder $\varepsilon_2 : \mathbb{R}^{n \times p} \to \mathbb{R}^{n_2 \times p}$ such that every column of $\varepsilon_2(X)$ contains a copy of $v_2(X)$ in the form*

$$\varepsilon_2(X) = \begin{bmatrix} v_2(X) & 0 & \cdots & 0 \\ 0 & v_2(X) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_2(X) \end{bmatrix} \in \mathbb{R}^{n_2 \times p}.$$

More precisely, there is a h_1-headed attention module $\alpha_1 : \mathbb{R}^{n \times p} \to \mathbb{R}^{mh_1 \times p}$, a one-layer neural network, $\varphi_1 : \mathbb{R}^{mh_1 \times p} \to \mathbb{R}^{n_1 \times p}$, a h_2-headed attention module $\alpha_2 : \mathbb{R}^{n_1 \times p} \to \mathbb{R}^{mh_2 \times p}$, and another one-layer neural network $\varphi_2 : \mathbb{R}^{mh_2 \times p} \to \mathbb{R}^{n_2 \times p}$, such that

$$\varepsilon_2 = \varphi_2 \circ \alpha_2 \circ \varphi_1 \circ \alpha_1. \tag{19}$$

In particular, any monomial of degree not more than two in the entries of X appears in every column of $\varepsilon_2(X)$.

Proof. We will first construct a multihead attention module α with the property that each of the p columns of $\alpha(X)$ contains every entry of X, i.e., x_{ij}, $i = 1, \dots, n$, $j = 1, \dots, p$. Fix any $(\hat{\imath}, \hat{\jmath}, j)$ and consider the single-head attention module $\alpha_{\hat{\imath}\hat{\jmath}j} : \mathbb{R}^{n \times p} \to \mathbb{R}^{m \times p}$ as in (3) with

$$A_V = E_{1\hat{\imath}}, \quad B_V = 0, \quad A_K = 0, \quad B_K = E_{1\hat{\jmath}}, \quad A_Q = 0, \quad B_Q = E_{1j},$$

as in (4). Then the $(1, j)$th entry of $\alpha_{\hat{\imath}\hat{\jmath}j}(X)$ is exactly $x_{\hat{\imath}\hat{\jmath}}$ and all other entries in the first row are zeros. If we repeat this for all $\hat{\imath} \in \{1, \dots, n\}$, $\hat{\jmath}, j \in \{1, \dots, p\}$ and stack these np^2 attention modules together, we obtain the multihead attention α. By construction any column of $\alpha(X)$ contains every entry of X.

For the required α_1, we need to augment α so that every column of $\alpha_1(X)$ will also contain the constant 1. Consider the single-head attention module $\alpha^{(j)} : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ with

$$A_V = 0, \quad B_V = E_{11}, \quad A_K = 0, \quad B_K = E_{11}, \quad A_Q = 0, \quad B_Q = E_{1j}.$$

Then the $(1, j)$th entry of $\alpha^{(j)}(X)$ is 1, and all other entries in the first row are zeros. We repeat this for all $j \in \{1, \dots, p\}$ and stack $\alpha^{(1)}, \dots, \alpha^{(p)}$ with α to obtain the required α_1. Note that α_1 has $h_1 = np^2 + p$ heads.

Because $x = \operatorname{ReLU}(x) - \operatorname{ReLU}(-x)$, the coordinate function $f(x_1, \dots, x_n) = x_i = \operatorname{ReLU}(x_i) - \operatorname{ReLU}(-x_i)$ can be represented using a one-layer neural network. So there exists a one-layer neural network φ_1 that only keeps all first rows of the above attention modules, and $\varepsilon_1 = \varphi_1 \circ \alpha_1$ gives

$$\varepsilon_1(X) = \begin{bmatrix} v_1(X) & 0 & \cdots & 0 \\ 0 & v_1(X) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_1(X) \end{bmatrix} \in \mathbb{R}^{n_1 \times p}.$$

We will now construct α_2. We first repeat the construction above so that the first $np^2 + p$ heads of α_2 will produce all linear monomials (i.e., the entries of X), and the constant. In particular, by the end of our construction, every column of $\alpha_2 \circ \varphi_1 \circ \alpha_1(X)$ will contain every entry of X and 1, and each of these entries is the only nonzero entry in its row.

We then construct the next batch of heads of α_2 that will produce all quadratic monomials. Consider the attention module $\alpha_{\hat{\imath}\bar{\imath}j}$ defined by

$$A_V = E_{1\hat{\imath}}, \quad B_V = 0, \quad A_K = 0, \quad B_K = E_{1j}, \quad A_Q = E_{1\bar{\imath}}, \quad B_Q = 0.$$

Then the $(1, j)$th entry of $\alpha_{\hat{\imath}\bar{\imath}j}(X)$ is $x_{\hat{\imath}j}(x_{\bar{\imath}j})^+$. In other words we can form quadratic terms in jth column out of entries in jth column. If we repeat this for all $\hat{\imath}, \bar{\imath} \in \{1, \dots, mh_1\}$, $j \in \{1, \dots, p\}$, and stack these attention modules together, we obtain a multihead attention α_2^+.

By our previous construction, among the rows of $\varphi_1 \circ \alpha_1(X)$ are two with only the jth column nonzero, taking values $x_{\hat{\imath}\hat{\jmath}}$ and $x_{\bar{\imath}\bar{\jmath}}$ respectively. Composing with α_2, we obtain a row with jth entry $x_{\hat{\imath}\hat{\jmath}}(x_{\bar{\imath}\bar{\jmath}})^+$, and other entries zeros. So the composition $\alpha_2^+ \circ \varphi_1 \circ \alpha_1(X)$ contains all quadratic terms of the form $x_{\hat{\imath}\hat{\jmath}}(x_{\bar{\imath}\bar{\jmath}})^+$ in any column, and each of those entries is the only nonzero entry in its row. We may repeat the same argument to obtain a multihead attention α_2^- with the property that the composition $\alpha_2^- \circ \varphi_1 \circ \alpha_1(X)$ contains all quadratic terms of the form $x_{\hat{\imath}\hat{\jmath}}(-x_{\bar{\imath}\bar{\jmath}})^+$ in any column, and each of those entries is the only nonzero entry in its row. The required α_2 is then obtained by stacking α_2^+, α_2^-, together with the first $np^2 + p$ heads that give the linear monomials and constant.

The one-layer neural network φ_2 is then chosen so that it gives the quadratic monomial

$$x_{\hat{\imath}\hat{\jmath}} x_{\bar{\imath}\bar{\jmath}} = x_{\hat{\imath}\hat{\jmath}}(x_{\bar{\imath}\bar{\jmath}})^+ - x_{\hat{\imath}\hat{\jmath}}(-x_{\bar{\imath}\bar{\jmath}})^+,$$

for $\hat{\imath}, \bar{\imath} \in \{1, \dots, n\}$ and $\hat{\jmath}, \bar{\jmath} \in \{1, \dots, p\}$. □

Recall from Section 1.2 that whenever a neural network takes a matrix input $X = [x_1, \dots, x_p] \in \mathbb{R}^{n\times p}$, it is applied columnwise to each column $x_j \in \mathbb{R}^n$. In general an attention module and a neural network are distinct objects. But there is one special case when they are related.

Lemma 3.7 (One-layer neural networks as encoder blocks)**.** *Let $\varphi : \mathbb{R}^n \to \mathbb{R}^{n_2}$ be a one-layer neural network. Then*

$$\varphi : \mathbb{R}^{n\times p} \to \mathbb{R}^{n_2\times p}, \quad [x_1, \dots, x_p] \mapsto [\varphi(x_1), \dots, \varphi(x_p)],$$

is an encoder block of the form $\varphi_1 \circ \alpha_1$ where φ_1 is also a one-layer neural network and α_1 is an attention module.

Proof. Consider the attention module $\alpha_{ij} : \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}$ given by

$$A_V = E_{1i}, \quad B_V = 0, \quad A_K = 0, \quad B_K = E_{1j}, \quad A_Q = 0, \quad B_Q = E_{1j}.$$

The first row of $\alpha_{ij}(X)$ has x_{ij} in its $(1, j)$th entry and zeros elsewhere. If we stack these attention modules α_{ij}, $i \in \{1, \dots, n\}$, $j \in \{1, \dots, p\}$ together, we obtain an np-headed attention module $\alpha : \mathbb{R}^{n\times p} \to \mathbb{R}^{mnp\times p}$. By construction, $\alpha(X)$ contains a submatrix of the form

$$\begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_p \end{bmatrix} \in \mathbb{R}^{np\times p}, \tag{20}$$

where $x_j \in \mathbb{R}^n$ is the jth column of $X \in \mathbb{R}^{n\times p}$, $j = 1, \dots, p$. Let $\psi : \mathbb{R}^{np} \to \mathbb{R}^p$ be the affine map given by

$$\mathbb{R}^{np} \ni \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \mapsto x_1 + x_2 + \cdots + x_p \in \mathbb{R}^n.$$

We apply ψ columnwise to $\alpha(X)$, extending its domain so that ψ maps every row outside the submatrix in (20) to zero. Then the submatrix in (20) is transformed as

$$\begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_p \end{bmatrix} \mapsto [x_1, x_2, \dots, x_p] = X$$

and every other row outside of this submatrix gets mapped to zero. In other words ψ is a left inverse of α. The required statement then follows from $(\varphi \circ \psi) \circ \alpha = \varphi \circ (\psi \circ \alpha) = \varphi$, with $\varphi_1 = \varphi \circ \psi$ and $\alpha_1 = \alpha$. □

While the definition of an encoder as in Section 1.4 does not require the neural networks within it to have only one hidden layer, the original version in [45] does. Lemma 3.7 shows that this is not really a more stringent requirement since whenever we are presented with a multilayer neural network we may repeatedly apply Lemma 3.7 to turn it into the form required in [45].

Assuming the Pierce–Birkhoff conjecture, we may now show that any matrix-valued spline $f : \mathbb{R}^{n\times p} \to \mathbb{R}^{r\times p}$ is an encoder. We prove the most general case possible so that other special cases follow effortlessly: the corresponding result for vector-valued splines is obtained by setting $p = 1$ and that for scalar-valued splines by setting $r = p = 1$. Note also that the result below applies to splines defined on any semialgebraic partition — the most common rectilinear partition obtained through triangular of domain is also a special case.

Theorem 3.8 (Splines as encoders)**.** *Let $f : \mathbb{R}^{n\times p} \to \mathbb{R}^{r\times p}$ be a max-definable function. Then f is a t-layer encoder for some finite $t \in \mathbb{N}$. More precisely, there exist t attention modules $\alpha_1, \dots, \alpha_t$ and t one-layer neural networks $\varphi_1, \dots, \varphi_t$ such that*

$$f = \varphi_t \circ \alpha_t \circ \varphi_{t-1} \circ \alpha_{t-1} \circ \cdots \circ \varphi_1 \circ \alpha_1.$$

If the Pierce–Birkhoff conjecture holds, then any degree-k spline is an encoder.

Proof. Let $f(X) = [f_1(X), \dots, f_p(X)] \in \mathbb{R}^{r\times p}$ with $f_j(X) \in \mathbb{R}^r$, $j = 1, \dots, p$. By Lemma 3.5, we may write $f_j = \ell_j \circ v_{k_j}$ where ℓ_j is a linear spline and v_{k_j} the Veronese map of degree k_j. Let $s := \max(k_1, \dots, k_p)$. Then by padding ℓ_j with extra terms with zero coefficients, we may assume

$$f_j = \ell_j \circ v_{k_j} = \ell_j \circ v_s. \tag{21}$$

Note that $\ell_j : \mathbb{R}^{\binom{np+s}{s}} \to \mathbb{R}^r$.

It follows from Lemma 3.4 that we may obtain all monomials of degree not more than s by composing the quadratic Veronese map with itself sufficiently many times. So by composing $\lceil \log_2 s \rceil$ copies of the encoder constructed in Lemma 3.6, we obtain an encoder

$$\varepsilon \coloneqq \varepsilon_2 \circ \cdots \circ \varepsilon_2 \tag{22}$$

with the property that any column of $\varepsilon(X)$ contains a copy of Veronese map of degree s, i.e.,

$$\varepsilon(X) = \begin{bmatrix} v_s(X) & 0 & \cdots & 0 \\ 0 & v_s(X) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_s(X) \end{bmatrix} \in \mathbb{R}^{n_t \times p}.$$

There is a slight abuse of notation in (22): We have assumed that the $(i+1)$th copy of ε_2 has input dimension $n_{i+1} \coloneqq (n_i p + 2)(n_i p + 1)/2$, the output dimension of the ith copy of ε_2. Strictly speaking these are different maps since domains and codomains are different although we denote all of them as ε_2. Also, in the final layer, we drop any rows that we do not need — this is not a problem as "dropping rows" is just a modification of the neural network in the last layer φ_t, which we will be modifying anyway below.

Expanding each copy of ε_2 as in (19), we obtain the structure in (7), i.e.,

$$\varepsilon = \varphi_t \circ \alpha_t \circ \varphi_{t-1} \circ \alpha_{t-1} \circ \cdots \circ \varphi_1 \circ \alpha_1 \tag{23}$$

for some $t \in \mathbb{N}$. We will modify the attention module $\alpha_t : \mathbb{R}^{n_t \times p} \to \mathbb{R}^{m_t \times p}$ in the last layer. For $i = 1, \dots, 2r$, we let $\alpha^{(i)} : \mathbb{R}^{n_t \times p} \to \mathbb{R}^{m \times p}$ be a (single-head) attention module with

$$A_V^{(i)} = 0, \quad A_K^{(i)} = 0, \quad A_Q^{(i)} = 0, \quad B_V^{(i)} = E_{11}, \quad B_K^{(i)} = E_{11},$$

and $B_Q^{(i)}$ is a nonnegative constant matrix to be determined later. The first row of $\alpha^{(i)}(X)$ is the first row of $B_Q^{(i)}$, i.e., $\alpha^{(i)}(X)$ contains a row of nonnegative constants. By stacking $2r$ heads $\alpha^{(1)}, \dots, \alpha^{(2r)}$ onto α_t, we obtain a modified attention module with $2r$ extra heads,

$$\widehat{\alpha}_t \coloneqq (\alpha^{(1)}, \dots, \alpha^{(2r)}, \alpha_t) : \mathbb{R}^{n_t \times p} \to \mathbb{R}^{(2mr + m_t) \times p}$$

Prefixing these heads to α_t will allow us to add $2r$ rows of nonnegative constants to $\varepsilon(X)$.

First, by modifying the neural network φ_t to $\widehat{\varphi}_t$, one that keeps the first row of each those $2r$ extra heads, we see that $\widehat{\varphi}_t \circ \widehat{\alpha}_t(X)$ will have $2r$ rows of nonnegative constants irrespective of X. We may also choose $\widehat{\varphi}_t$ so that these occur as the first through $2r$th rows, denoted as

$$\begin{bmatrix} b_1 & b_2 & \cdots & b_p \\ b_1' & b_2' & \cdots & b_p' \end{bmatrix} \in \mathbb{R}^{2r \times p},$$

for some $b_i, b_i' \in \mathbb{R}_+^r$, $i = 1, \dots, p$. Note that each row of the matrix above comes from one of the added heads $\alpha^{(1)}, \dots, \alpha^{(2r)}$.

By replacing φ_t and α_t in (23) with $\widehat{\varphi}_t$ and $\widehat{\alpha}_t$, we obtain an encoder $\widehat{\varepsilon} : \mathbb{R}^{n \times p} \to \mathbb{R}^{(2r + n_t) \times p}$,

$$\widehat{\varepsilon} \coloneqq \widehat{\varphi}_t \circ \widehat{\alpha}_t \circ \varphi_{t-1} \circ \alpha_{t-1} \circ \cdots \circ \varphi_1 \circ \alpha_1.$$

By our construction we must have

$$\widehat{\varepsilon}(X) = \begin{bmatrix} b_1 & b_2 & \cdots & b_p \\ b_1' & b_2' & \cdots & b_p' \\ v_s(X) & 0 & \cdots & 0 \\ 0 & v_s(X) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_s(X) \end{bmatrix} \in \mathbb{R}^{(2r + n_t) \times p}.$$

Define the linear spline $\ell : \mathbb{R}^{2r+\binom{np+s}{s}p} \to \mathbb{R}^{(p+2)r}$ by

$$\ell(x, x', x_1, \dots, x_p) = \bigl(x, x', \ell_1(x_1), \dots, \ell_p(x_p)\bigr).$$

Here $x, x' \in \mathbb{R}^r$, $x_1, \dots, x_p \in \mathbb{R}^{\binom{np+s}{s}}$. By Theorem 3.1, linear splines are exactly neural networks. Recall from Section 1.2 that when we apply a neural network to a matrix, we apply it columnwise. Hence

$$\ell \circ \widehat{\varepsilon}(X) = \begin{bmatrix} b_1 & b_2 & \cdots & b_p \\ b_1' & b_2' & \cdots & b_p' \\ f_1(X) & \ell_1(0) & \cdots & \ell_1(0) \\ \ell_2(0) & f_2(X) & \cdots & \ell_2(0) \\ \vdots & \vdots & \ddots & \vdots \\ \ell_p(0) & \ell_p(0) & \cdots & f_p(X) \end{bmatrix},$$

where we have used (21). Now we set

$$b_i = \operatorname{ReLU}\Bigl(-\sum_{j \neq i} \ell_j(0)\Bigr), \quad b_i' = \operatorname{ReLU}\Bigl(\sum_{j \neq i} \ell_j(0)\Bigr),$$

for each $i = 1, \dots, p$. Let $\psi : \mathbb{R}^{(p+2)r} \to \mathbb{R}^r$ be the linear map defined by

$$\psi(y, y', y_1, \dots, y_p) = y - y' + y_1 + \dots + y_p,$$

where $y, y', y_1, \dots, y_p \in \mathbb{R}^r$. Then the composition of $\psi \circ \ell \circ \widehat{\varepsilon}$ has

$$\psi \circ \ell \circ \widehat{\varepsilon}(X) = [f_1(X), \dots, f_p(X)] = f(X),$$

as required. At this point we have obtained f as an encoder according to the definition in Section 1.4 since $\psi \circ \ell \circ \widehat{\varphi}_t$ is clearly a multilayer neural network. By our remark after the proof of Lemma 3.7, it may be converted into an alternate composition of attention modules and single-layer neural networks. □

In case the reader is wondering the value of s in the proof above is not necessarily k and can be strictly larger. To the best of our knowledge, there is not even a *conjectural* effective version of Conjecture 2.3 in the literature. So unlike Theorem 3.1, any bounds on the number of encoder blocks, number of heads of attention modules, width of the neural networks, etc, are beyond reach at this point.

Just as Theorem 3.1 establishes the equivalence between ReLU-neural networks and linear splines, various parts of the results in this article collectively establish the equivalence between ReLU-encoders and splines, assuming the validity of the Pierce–Birkhoff conjecture.

Corollary 3.9. *If the Pierce–Birkhoff conjecture holds, then the following classes of functions are all equal:*

(i) *splines;*
(ii) *encoders;*
(iii) *max-definable functions;*
(iv) *linear splines composed with the Veronese map.*

While our article is about understanding transformers in terms of splines, there is a somewhat unexpected payoff: the proof of Theorem 3.8 yields a way to construct autoregressive splines. There appears to be no universally agreed-upon meaning for the term "autoregressive spline" in the existing literature. In particular none replicates (8) and we are unaware of any construction that yields a spline that is autoregressive in the sense of (8).

Corollary 3.10 (Autoregressive splines as decoders)**.** *Let $k \in \mathbb{N}$ and $f : \mathbb{R}^{n \times p} \to \mathbb{R}^{m \times p}$ be an autoregressive max-definable function. Then f is a t-layer decoder for some finite $t \in \mathbb{N}$. More precisely, there exist t masked attention modules $\beta_1, \dots, \beta_t$ and t one-layer neural networks $\varphi_1, \dots, \varphi_t$ such that*

$$f = \varphi_t \circ \beta_t \circ \varphi_{t-1} \circ \beta_{t-1} \circ \cdots \circ \varphi_1 \circ \beta_1.$$

If the Pierce–Birkhoff conjecture holds, then any degree-k autoregressive spline is a decoder.

Proof. The proof of Lemma 3.6 applies almost verbatim. In fact it is slightly simpler since in the $(1, j)$th entry, we only need to construct monomials of the form $x_{\hat{\imath}\hat{\jmath}}$, $x_{\hat{\imath}\hat{\jmath}} x_{\bar{\imath}\bar{\jmath}}$ for $\hat{\jmath}, \bar{\jmath} \le j$. The same constructions used to obtain $A_V, B_V, A_K, B_K, A_Q, B_Q$ produce these required monomials when we use masked attention modules in place of attention modules. The proofs of Lemma 3.7 and Theorem 3.8 then apply with masked attention modules in place of attention modules. □

A similar construction can be extended to construct partially autoregressive splines as encoder–decoders.

4. Conclusion

It is an old refrain in mathematics that one does not really understand a mathematical proof until one can see how every step is inevitable. This is the level of understanding that we hope Section 3.3 provides for the transformer.

4.1. Insights.

Arora et al. [1] have shown that neural networks are exactly linear splines. Since compositions of linear splines are again linear splines, to obtain more complex functions we need something in addition to neural networks. Viewed in this manner, the attention module in Section 1.3 is the simplest function that serves the role. Lemma 3.6 shows that the quadratic Veronese map, arguably the simplest map that is not a linear spline, can be obtained by composing two attention modules. The proof reveals how heads and layers are essential: It would fail if we lacked the flexibility of having multiple heads and layers. The proof also shows how a neural network works hand-in-glove with attention module: It would again fail if we lack either one. The proof of Theorem 3.8 then builds on Lemma 3.6: By composing quadratic Veronese maps we can obtain Veronese map of any higher degree; and by further composing it with linear splines we obtain all possible splines. The resulting map, an alternating composition of attention modules and neural networks, is exactly the encoder of a transformer.

There are some other insights worth highlighting. Lemma 3.7 explains why the neural networks within a transformer require no more than one hidden layer; Vaswani et al. [45] likely arrived at this same conclusion through their experimentation. Theorem 3.3(vii) shows why layering attention modules and neural networks makes for an effective way to increase model complexity — the degree of the spline 3^t increases exponentially with the number of layers t.

4.2. Recommendations.

Recent work of Wortsman et al. [48] shows that a ReLU-transformer is perfectly capable of achieving results of similar quality as the original SoftMax-transformer, offering significant computational savings. We also advocate the use of ReLU activation, if only for turning a nearly-mystical and sometimes-feared technology into a familiar friendly one. In which case we could drop the word "smoothed" in our title — attention is a cubic spline.

If a smooth function is desired, we argue for using SoftPlus instead of SoftMax as activation. The SoftMax function is the natural smooth proxy for argmax as well as the derivative of SoftPlus, also known as the log-sum-exp function, which is in turn the natural smooth proxy for ReLU. Indeed SoftPlus has been used in place of ReLU to construct smooth neural networks with encouraging results [7]. Despite their intimate relationship, SoftMax makes for a poor proxy for ReLU. On the basis of our work, a SoftPlus-activation would be natural, smooth, and preserves fidelity with splines.

Lastly, Section 3.3 points to the importance of a nearly forgotten seventy-year-old conjecture about splines by one of its pioneers. Indeed, Theorem 3.8 shows that the Pierce–Birkhoff conjecture is true if and only if every spline is an encoder. Perhaps this article will rekindle interest in the conjecture and point a way towards its resolution.

References

[1] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

[3] Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023.

[4] G. Birkhoff and H. L. Garabedian. Smooth surface interpolation. *J. Math. and Phys.*, 39:258–268, 1960.

[5] G. Birkhoff and R. S. Pierce. Lattice-ordered rings. *An. Acad. Brasil. Ci.*, 28:41–69, 1956.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. volume 33, pages 1877–1901, 2020.

[7] G. C. Calafiore, S. Gaubert, and C. Possieri. Log-sum-exp neural networks and posynomial models for convex and log-log-convex data. *IEEE Trans. Neural Netw. Learn. Syst.*, 31(3):827–838, 2020.

[8] C. K. Chui. *Multivariate splines*, volume 54 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[9] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, 45(5):485–560, 1992.

[10] H. B. Curry and I. J. Schoenberg. On Pólya frequency functions. IV. The fundamental spline functions and their limits. *J. Analyse Math.*, 17:71–107, 1966.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2(4):303–314, 1989.

[12] C. de Boor. Splines as linear combinations of B-splines. A survey. In *Approximation theory, II (Proc. Internat. Sympos., Univ. Texas, Austin, Tex., 1976)*, pages 1–47. Academic Press, New York-London, 1976.

[13] C. de Boor. *A practical guide to splines*, volume 27 of *Applied Mathematical Sciences*. Springer-Verlag, New York-Berlin, 1978.

[14] C. de Boor. The way things were in multivariate splines: a personal view. In *Multiscale, nonlinear and adaptive approximation*, pages 19–37. Springer, Berlin, 2009.

[15] C. de Boor, K. Höllig, and S. Riemenschneider. *Box splines*, volume 98 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1993.

[16] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186. Association for Computational Linguistics, 2019.

[17] M. DiPasquale and F. Sottile. Bivariate semialgebraic splines. *J. Approx. Theory*, 254:105392, 19, 2020.

[18] M. DiPasquale, F. Sottile, and L. Sun. Semialgebraic splines. *Comput. Aided Geom. Design*, 55:29–47, 2017.

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

[20] G. Farin, J. Hoschek, and M.-S. Kim, editors. *Handbook of computer aided geometric design*. North-Holland, Amsterdam, 2002.

[21] J. Harris. *Algebraic geometry*, volume 133 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995.

[22] B. He, J. Martens, G. Zhang, A. Botev, A. Brock, S. L. Smith, and Y. W. Teh. Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation. In *International Conference on Learning Representations*, 2023.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[24] M. Henriksen and J. R. Isbell. Lattice-ordered rings and function rings. *Pacific J. Math.*, 12:533–565, 1962.

[25] P. Kidger and T. Lyons. Universal approximation with deep narrow networks. In *Conference on Learning Theory*, pages 2306–2327, 2020.

[26] D. E. Knuth. *The Metafont book*. Addison-Wesley, Boston, MA, 1989.

[27] S. A. Koohpayegani and H. Pirsiavash. Sima: Simple softmax-free attention for vision transformers. In *IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2607–2617, 2024.

[28] J. B. Lasserre. *An introduction to polynomial and semi-algebraic optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2015.

[29] L.-H. Lim. Tensors in computations. *Acta Numer.*, 30:555–764, 2021.

[30] J. Lu, J. Yao, J. Zhang, X. Zhu, H. Xu, W. Gao, C. XU, T. Xiang, and L. Zhang. SOFT: softmax-free transformer with linear complexity. In *Advances in Neural Information Processing Systems*, volume 34, pages 21297–21309, 2021.

[31] F. Lucas, D. Schaub, and M. Spivakovsky. On the Pierce-Birkhoff conjecture. *J. Algebra*, 435:124–158, 2015.

[32] D. Maclagan and B. Sturmfels. *Introduction to tropical geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2015.

[33] L. Mahé. On the Pierce-Birkhoff conjecture. *Rocky Mountain J. Math.*, 14(4):983–985, 1984.

[34] M. Marshall. The Pierce-Birkhoff conjecture for curves. *Canad. J. Math.*, 44(6):1262–1271, 1992.

[35] G. Micula and S. Micula. *Handbook of splines*, volume 462 of *Mathematics and its Applications*. Kluwer Academic Publishers, Dordrecht, 1999.

[36] S. Ovchinnikov. Max-min representation of piecewise linear functions. *Beiträge Algebra Geom.*, 43(1):297–302, 2002.

[37] B. Peters, V. Niculae, and A. F. Martins. Sparse sequence-to-sequence models. In *Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519, 2019.

[38] A. Prakash, K. Chitta, and A. Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7077–7087, 2021.

[39] Z. Qin, W. Sun, H. Deng, D. Li, Y. Wei, B. Lv, J. Yan, L. Kong, and Y. Zhong. cosFormer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2021.

[40] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.

[41] I. J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. Part A. On the problem of smoothing or graduation. A first class of analytic approximation formulae. *Quart. Appl. Math.*, 4:45–99, 1946.

[42] I. J. Schoenberg. Spline functions and the problem of graduation. *Proc. Nat. Acad. Sci. U.S.A.*, 52:947–950, 1964.

[43] B. Shekhtman and T. Sorokina. A note on intrinsic supersmoothness of bivariate semialgebraic splines. *Comput. Aided Geom. Design*, 98:Paper No. 102137, 5, 2022.

[44] E. V. Shikin and A. I. Plis. *Handbook on splines for the user*. CRC Press, Boca Raton, FL, 1995.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[46] S. Wagner. On the Pierce-Birkhoff conjecture for smooth affine surfaces over real closed fields. *Ann. Fac. Sci. Toulouse Math. (6)*, 19:221–242, 2010.

[47] G. Wahba. *Spline models for observational data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[48] M. Wortsman, J. Lee, J. Gilmer, and S. Kornblith. Replacing softmax with ReLU in vision transformers. *arXiv:2309.08586*, 2023.

[49] D. M. Young. Garrett Birkhoff and applied mathematics. *Notices Amer. Math. Soc.*, 44(11):1446–1450, 1997.

[50] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, Q. Vuong, V. Vanhoucke, H. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sanketi, G. Salazar, M. S. Ryoo, K. Reymann, K. Rao, K. Pertsch, I. Mordatch, H. Michalewski, Y. Lu, S. Levine, L. Lee, T.-W. E. Lee, I. Leal, Y. Kuang, D. Kalashnikov, R. Julian, N. J. Joshi, A. Irpan, B. Ichter, J. Hsu, A. Herzog, K. Hausman, K. Gopalakrishnan, C. Fu, P. Florence, C. Finn, K. A. Dubey, D. Driess, T. Ding, K. M. Choromanski, X. Chen, Y. Chebotar, J. Carbajal, N. Brown, A. Brohan, M. G. Arenas, and K. Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, volume 229, pages 2165–2183, 2023.

Department of Mathematics, University of Texas, Austin, TX 78712
Email address: zehua.lai@austin.utexas.edu

Computational and Applied Mathematics Initiative, Department of Statistics, University of Chicago, Chicago, IL 60637
Email address: lekheng@uchicago.edu

School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332
Email address: yucongliu@gatech.edu