Universal Approximation Properties of Neural Networks and Transformers

Venkata Hasith Vattikuti, vv8379

April 25, 2025

Contents

1 Introduction

Transformers have been at the heart of the AI boom due to them being the main mechanism behind many generative AI models such as the generative pre-trained transformer (which is the namesake of GPT), the diffusion transformer, and even more specialized problems like protein folding.

Since they have been very successful at a wide range of tasks, it is interesting to explore their universal approximation properties in order to understand the classes of functions that transformers are restricted to exploring and what kinds of transformer architectures can guarantee arbitrarily high accuracy in approximating them.

This report has three main goals. First, we will define a basic artificial neural network and explore their universal approximation properties. Then, we will construct a standard transformer model to understand the mappings that they define. Once we understand how transformers work, we will review recent literature on their universal approximation properties.

After reading this report, one will have a firm grasp on the structure of artificial neural networks and transformers and also understand some recent advances in their approximation properties.

2 Notation

- $I_n = [0,1]^n$
- $\mathbb{N} = \{1, 2, 3, \ldots\}$
- $[n] = \{1, 2, \ldots, n\}$, for $n \in \mathbb{N}$,
- $w_{[n]} = \{w_1, w_2, \ldots, w_n\}$
- $\mathbf{1}_n = [1, 1, \ldots, 1] \in \mathbb{R}^n$
- $\mathbf{1}_{m\times n} = \mathbf{1}_m \cdot \mathbf{1}_n^\top \in \mathbb{R}^{m\times n}$
- $U(S) = \{s \in S | \|s\| < 1\}$ (the norm will be the standard norm for that set unless otherwise specified).

3 Preliminaries

3.1 Norms and Metric Spaces

It will be important to have a formal way to describe the 'sizes' of elements in vector spaces, which we can do through the concept of norms.

Definition 1 *For a vector space X, a norm is a mapping*

$$\|\cdot\| : X \to [0, \infty)$$

that satisfies the following conditions for any $x, y \in X$ and $\lambda \in \mathbb{R}$:

- $\|\lambda x\| = \lambda \|x\|$,
- $\|x\| = 0 \iff x = 0$,
- $\|x + y\| \leq \|x\| + \|y\|$.

For any norm $\|\cdot\|$ on a vector space X, we have an induced metric of $d_X(x, y) = \|x - y\|$ for any $x, y \in X$.

When dealing with Euclidean vectors, we will by default be using the Euclidean norm as stated in Definition 2.

Definition 2 *The Euclidean norm for any $x \in \mathbb{R}^m$ for any $m \in \mathbb{N}$ is given by*

$$\|x\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2},$$

where x_i is the i^{th} component of the vector x. The Euclidean norm will be assumed for any vector unless otherwise stated, and will thus also be denoted simply as $\|x\|$ sometimes, where the fact that x is a Euclidean vector will be sufficient to assume that we are using the Euclidean norm.

We can generalize the concept of a Euclidean norm to a finite-dimensional vector norm.

Definition 3 *A vector norm or any $x \in \mathbb{R}^m$ for any $m \in \mathbb{N}$ is given by*

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{m} x_i^p}.$$

For vectors of infinite dimensions (also known as sequences), we use the ℓ^p norm which is a simple extension of Definition 3.

Definition 4 *For a sequence $(x_1, x_2, x_3, \ldots)$, the ℓ^p norm, for $1 \leq p < \infty$ is defined as*

$$\|x\|_p = (|x_1|^p + |x_2|^p + |x_3|^p + \ldots)^{1/p}.$$

To help us relate spaces to each other, it will also be important to understand the concept of one set of functions being dense in another set.

Definition 5 *For some set X, an associated metric d on the set, and two subsets $Y, Z \subset X$, we say Y is dense in Z if, for all $z \in Z$ and any $\epsilon > 0$, there exists some $y \in Y$ such that*

$$d(y, z) < \epsilon.$$

The concept of some set of functions Y being dense in another set Z will help us understand how well one class of functions approximates another class because it allows us to say that for any choice of function in Z, there will always exist some function in Y that is arbitrarily close to it under the metric d.

We will always assume that we are using the norm-induced metric space for a vector space, but the case of the Euclidean metric appears frequently enough that it is worth mentioning. When dealing with Euclidean spaces, we will always assume the Euclidean metric as seen in Definition 6

Definition 6 *The Euclidean metric for two vectors $x, y \in \mathbb{R}^m$ is defined as*

$$d(\mathbf{x}, \mathbf{y}) = \|x - y\|_2$$

where we made use of Definition 2 to define the Euclidean metric as the Euclidean norm of the difference between the two vectors.

Because it is used so often, anytime we deal with vectors or scalars, $|\cdot|$ or $\|\cdot\|$ will be referring to the Euclidean metric.

3.2 Important Classes of Functions

When studying the approximation properties of certain functions (in this case, neural networks which we will define in Section 4), we need a formal way to talk about certain classes of functions.

3.2.1 Continuous Functions

One of the most familiar classes of functions are continuous functions.

Definition 7 *A real function $f : X \to Y$ is continuous at some $x_0 \in X$ under metrics d_X and d_Y for the domain and codomain, respectively if for any $\epsilon > 0$, there exists some $\delta > 0$ such that*

$$d_X(x_0, x_1) < \delta \Rightarrow d_Y(f(x_0), f(x_1)) < \epsilon.$$

Definition 8 *A function $f : X \to Y$ is said to be continuous if f is continuous at all $x \in X$.*

In addition to continuous functions, another class of functions that we will encounter are functions with compact support on some domain.

Definition 9 *A function $f : X \to Y$ has compact support on some domain $\Omega \subset X$ if Ω is compact and*

$$f(x) = 0, \quad \forall\, x \notin \Omega$$

We will be particularly interested in real functions $f : X \to \mathbb{R}$, and the metric on any subset of $\mathbb{R}$ will always be assumed to be the Euclidean metric unless otherwise stated. We will the denote the set of all real continuous functions on X as $C(X)$:

$$C(X) = \{f : X \to \mathbb{R} | f \text{ is continuous}\}.$$

Additionally, we can define a simple norm on $C(X)$ as

$$\|f\|_{C(X)} = \sup_{x \in X} |f(x)|.$$

3.2.2 Lebesgue Integrable Functions

Another important class of functions are Lebesgue integrable functions. To define Lebesgue integrable functions, we must first understand the concept of the Lebesgue measure and Lebesgue-measurable functions which can be found in [1].

Definition 10 *For a measurable set $\Omega \subset \mathbb{R}^m$, a function $f : \Omega \to \overline{\mathbb{R}}$ is measurable if the inverse image of every open set in $\mathbb{R}$ is measurable.*

Definition 11 *A real measurable function f is integrable over Ω if*

$$\int_\Omega |f(x)| \mu(dx) < \infty.$$

The set of all Lebesgue integrable functions on some domain Ω is denoted as

$$\mathcal{L}(\Omega) = \left\{ f : \int_\Omega |f(x)| \mu(dx) < \infty \right\}.$$

3.2.3 Lebesgue Spaces

Lebesgue spaces, also called L^p spaces are defined for $1 \leq p < \infty$.

Definition 12 *For some measurable set $\Omega \subset \mathbb{R}^m$ and a p such that $0 < p < \infty$, we denote $L^p(\Omega)$ as the set of all measurable functions $f : \Omega \to \mathbb{R}$ such that*

$$\int_\Omega |f(x)|^p \mu(dx) < \infty.$$

Note that Definition 11 simply comes from the special case of an L^p space where $p = 1$.

Each $L^p(\Omega)$ space also admits a norm [1], denoted by $\|\cdot\|_{L^p(\Omega)}$, where Ω is the domain that the L^p space is defined on. The exact form of the norm for $1 < p < \infty$ is given by

$$\|f\|_{L^p(\Omega)} = \left(\int_\Omega |f(x)|^p \mu(dx)\right)^{1/p}, \quad \forall\, f \in L^p(\Omega)$$

and we may simply refer to this as $\|f\|_p$ for brevity when it is clear that f occupies the L^p space on some domain Ω.

In our brief overview of L^p spaces, the $p = \infty$ case was purposefully omitted due to its limited use in future sections, but the exact definitions of L^∞ and its corresponding norm can be found in [1].

3.2.4 Sobolev Spaces

To classify continuous functions based on smoothness, we need to refer to how many derivatives we can take. For some domain Ω, let $C^r(\Omega)$ be the set of all functions with well defined derivatives $D^\alpha f$ for all α, $|\alpha| = \sum_i^d |\alpha_i| = 1$; the norm for $C^r(\Omega)$ is given by

$$\|f\|_{C^r(\Omega)} = \max_{|\alpha|=r} \|D^\alpha f\|_{C(\Omega)} + \|f\|_{C(\Omega)}.$$

Then, Sobolev spaces spaces generalize C^r spaces by allowing for weak derivatives [2] in $D^\alpha f$; if we allow such a broadening of our requirement for D^α, we can define the Sobolev space $W^{r,p}(\Omega)$ or $W^r(L^p(\Omega))$ as the set of all $f \in L^p(\Omega)$ for any $1 \le p \le \infty$ such that $D^\alpha f \in L^p(\Omega)$ for all $|\alpha| = r$. We prescribe Sobolev spaces with a norm similar to that of C^r:

$$\|f\|_{W^r(L^p(\Omega))} = \max_{|\alpha|=r} \|D^\alpha f\|_{L^p(\Omega)} + \|f\|_{L^p(\Omega)}.$$

3.2.5 Besov Spaces

Besov spaces generalize C^r one step further by allowing for non-integer classes of smoothness [3]. To define Besov spaces, we first define a modulus of smoothness of order r for a function $f \in L_p(\Omega), 0 < p \le \infty$ as

$$\omega_r(f,t)_p = \sup_{0<|h|\le t} \|\Delta_h^r(f,\cdot)\|_{L_p(\Omega)}, \quad t > 0,$$

where $h \in \mathbb{R}^d$, and

$$\Delta_h^r(f,x) = \sum_{k=0}^{r} (-1)^{r-k} \binom{r}{k} f(x+kh), \quad x \in \Omega \subset \mathbb{R}^d$$

so that $\omega_r(f,t)_p \to 0$ as $t \to 0$ and the rate of convergence depends on the smoothness of f on $L^p(\Omega)$.

To allow for non-integer values of r, Besov spaces replace r with any $s > 0$ and introduce a new parameter $0 < q \leq \infty$. The Besov space $B_q^s(L^p(\Omega))$ is defined as the set of functions $f \in L^p(\Omega)$ for which

$$|f|_{B_q^s(L_p(\Omega))} = \|t^{-s}\omega_r(f,t)_p\|_{L^q((0,\infty),\mu(dx)/x)} < \infty$$

where $L^q((0,\infty),\mu(dx)/x)$ refers to the L^p space on $(0,\infty)$ with respect to the measure $\mu(dx)/x$ rather than just $\mu(dx)$. The norm on such a Besov space is

$$\|f\|_{B_q^s(L^p(\Omega))} = |f|_{B_q^s(L_p(\Omega))} + \|f\|_{L^p(\Omega)}.$$

3.2.6 Lipschitz Spaces

Another useful space is the Lipschitz space $\mathrm{Lip}(\alpha, p)$ for $0 < \alpha \leq 1$ and $0 < p \leq \infty$ consisting of functions $f \in L^p(\Omega)$ for which

$$\omega_1(f,t)_p \leq Mt^\alpha.$$

The $p = \infty$ case is particularly important so we will define $\mathrm{Lip}\ \alpha = \mathrm{Lip}(\alpha, C(\Omega))$ for some domain Ω that should be clear from context if not explicitly stated.

4 Artificial Neural Networks

Artificial neural networks (ANNs), commonly referred to as neural networks or feed-forward networks, are machine learning models inspired by the human brain [4]. The most basic elements of a neural network are nodes known as 'neurons', and they pass information through series of neurons known as 'layers' with linear and nonlinear transformations [5].

4.1 The Feed Forward Network

Concretely, a neuron is a function [6] $y : \mathbb{R}^m \to \mathbb{R}$ that is parameterized by a set of weights $\{w_i \in \mathbb{R}\}_{i\in[m]}$, a bias $b \in \mathbb{R}$, and an activation function $\phi : \mathbb{R} \to \mathbb{R}$. The formula used to compute the output of y for an input $x \in \mathbb{R}^m$ is given as

$$y(x) = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right), \tag{1}$$

with x_i denoting the i-th component of the vector x. Visually, this can be represented as in Figure 1 where each input is multiplied by a corresponding weight parameter, summed across all inputs, and then passed into an activation function with an added bias. For a layer made up of n neurons $\{y_1, y_2, \ldots, y_n\}$, we can define a vector valued function [8] $h : \mathbb{R}^m \to \mathbb{R}^n$, where the i-th component is given by

$$\tilde{h}_i(x) = \sum_{j=1}^{m} w_{ij} x_j + b_i, \tag{2}$$

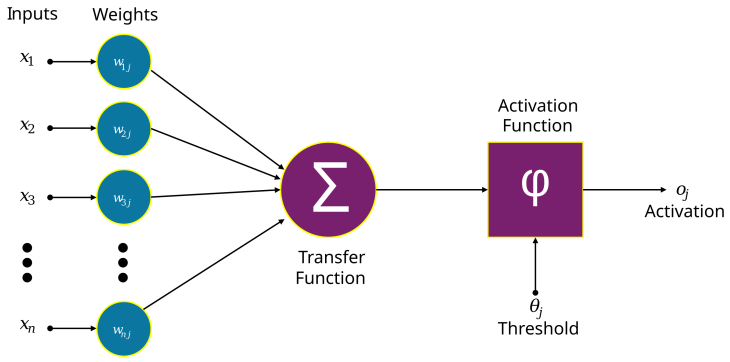

Figure 1: A visualization of a single neuron with an n-dimensional input x, weights $w_{[n],j}$, a bias θ_j (referred to as a threshold in the figure), and an activation function $\varphi : \mathbb{R} \to \mathbb{R}$. The j index simply denotes the index of the neuron itself to remind us of the fact that the neuron is one part of a larger network. Each component of the input x_i gets multiplied by its corresponding weight w_{ij}, then we obtain the sum $s = \Sigma_i x_i w_{ij}$, and pass it into the activation function with a bias to get an output activation: $o_j = \varphi(s + \theta_j)$. Image courtesy of Wikipedia [7].

$$h_i(x) = \phi\left(\tilde{h}_i(x)\right), \tag{3}$$

where w_{ij} and b_i denote the weight w_j and the bias b of the i-th neuron in the language of Equation (1). Note that Equation (3) is simply the output of the i-th neuron, $h_i(x) = y_i(x)$.

In the interest of being more concise, we can write Equation (1) using a vector to represent the weights, $\alpha \in \mathbb{R}^n$. Then, we have

$$y(x) = \phi(\alpha^\top x + b) \tag{4}$$

Additionally, we can also express the full form of $\tilde{h}$ in a compact way by using matrices to represent the linear transformation of multiplying and summing the weights with the input:

$$\tilde{h}(x) = Wx + b \tag{5}$$

$$h(x) = \phi(\tilde{h}(x)) \tag{6}$$

Here, in Equation (5), all the weights have been collected into the term $W \in \mathbb{R}^{n \times m}$ with the each i, j element being w_{ij}, all the biases have been collected into $b \in \mathbb{R}^m$ with the i-th component being equal to b_i, and the activation function in Equation (6) ϕ is applied element wise. The function h is said to represent a layer of neurons and the output, $h(x)$, is the output of that layer for an input x.

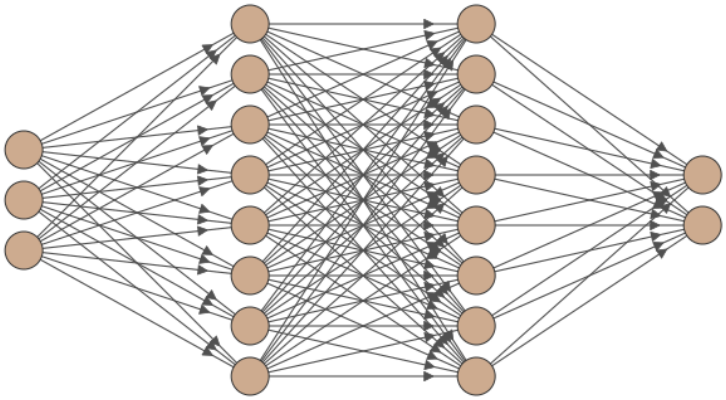

Figure 2: A feed-forward network with a 3-dimensional input, a 2-dimensional output, and 8-dimensional hidden layers. The edges indicate a weight between one node and another, and the arrows show the direction of computation of the network. Image courtesy of Alexander LeNail [9].

An neural network $f : \mathbb{R}^m \to \mathbb{R}^m$ is defined as a composition of L layers of neurons [5],

$$f(x) = [\tilde{h}^L \circ h^{L-1} \circ \dots h^1](x), \tag{7}$$

where $h^k : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k-1}}$ for $k \in [L]$ is the function corresponding to the k-th layer of the neural network, and d_k is the number of neurons in the k-th layer, with $d_0 = m$ and $d_L = n$. As an aside, the layers besides the the last layer (those denoted by the indices $1, 2, \dots, L-1$) are often referred to as hidden layers, and the number of hidden layers are often times simply referred to as the number of layers since we are always guaranteed to have an input and output layer anyway. Figure 2 depicts a nerual network made up of two composed functions–meaning two hidden layers.

Due to the composition of multiple layers each with multiple neurons, f is a function parametrized by the weights of each layer $W^{[L]}$ and the biases $b^{[L]}$. Note that each $d_{[L]}$, the number of neurons in each layer; $\phi^{[L]}$, the choice of activation functions; as well as L itself, the number of layers, are not usually treated as hyperparameters are instead treated as constants. Colloquially, each $d_{[L]}$ is sometimes called the width of the network at each layer, and L is known as the depth of the network.

Rather than tediously keep track of every parameter as we discuss neural networks, it is commonplace to gather every hyperparameter into one term θ, and–to remind ourselves that f is dependent upon these–denote f as $f(\cdot; \theta)$ or f_θ, and often times θ is simply referred to as the weights of the model instead of just $w_{[d]}$ (and that convention will be used throughout this text). Collectively, the weights θ along with the choice of $d_{[L]}$ make up the architecture of the neural network. Architecture, however, is a loose term and generally refers to the

parameters and constants that specify how to compute the function f, extending even to other types of models that are not solely made up of neurons such as convolutional neural network (CNN) models [10], long short term memory (LSTM) models [11], transformer models [12], etc.

4.2 Universal Approximation Theorems of Neural Networks

Under various assumptions on the architectures of neural networks (such as specific activation functions, upper or lower bounds on the number of neurons, and a minimum number number of layers), multiple universal approximation theorems have been proven [3] which–in a qualitative sense–state that that for any function g in a certain class, there exists a set of weights θ such that f_θ can arbitrarily well approximate g, where f_θ is a neural network parametrized by θ.

In this section, we will explore some of these approximation theorems as they will be useful in understanding what kinds of functions transformer models are able to approximate. The cases we will be going over include the arbitrary width but bounded depth case, the bounded width but arbitrary depth case, and finally the bounded depth and width case for neural networks using the ReLU activation function,

$$\mathrm{ReLU}(x) = \max(x, 0).$$

First, to define the set of neural networks we will be studying, we will denote the class of functions corresponding ReLU networks of width W (meaning each layer has W neurons), depth L, an input of dimension d, and an output of dimension d' with $\Gamma^{W,L}(\mathrm{ReLU}; d, d')$; and each $\gamma \in \Gamma$ is determined by a total of

$$n(W, L) = (d + 1)W + W(W + 1)(L - 1) + d'(W + 1)$$

parameters (all the weights and biases of the network). Since we will assume the use of the ReLU activation and $d' = 1$ unless otherwise stated, we can use the shorthand $\Gamma^{W,L}$ when the role of d is clear from context.

Then, for some target function $f \in L^p(\Omega)$, we will be interested in the error of approximation

$$E(f, \Sigma_n)_{L_p(\Omega)} = E_n(f, \Sigma)_p = \inf_{s \in \Sigma_n} \|f - S\|_{L_p(\Omega)}$$

where Σ_n is a subset of $\Gamma^{W,L}$ with $n(W, L)$ parameters. The performance of a set of networks on some $K \subset L_p(\Omega)$ is given by

$$E(K, \Sigma_n)_p = E_n(K, \Sigma)_p = \sup_{f \in K} E(f, \Sigma_n)_p.$$

4.2.1 Single Layer ReLU Networks

The most simple case of neural networks that we can study are networks with a scalar input and output (meaning the neural network defines a one-dimensional scalar function) with a singular hidden layer.

Theorem 1 from [3] provides a theoretical guarantee for the approximation error for one-layer ReLU networks with input and output dimensions $d = d' = 1$, defined by the set $\Gamma^{n,1}$(ReLU;1,1).

Theorem 1 *Let $K = U(B_q^s(L_\tau(\Omega)))$ be the unit ball of the Besov space $B_q^s(L_\tau(\Omega))$ for $0 < \tau \leq \infty$. If $0 < s \leq 2$ and $s > \frac{1}{\tau} - \frac{1}{p}$ for $1 \leq p \leq \infty$, then*

$$E_n(K, \Sigma)_p \leq C(s, p, \tau)(n+1)^{-s}, \quad n \geq 0,$$

where $C(s, p, \tau)$ is some positive constant that is a function of s, p, τ. Theorem 1 shows us that increasing the smoothness exponent of the Besov ball, s, and the number of parameters in the network, n, will decrease the approximation error with a rate of $O((n+1)^{-s})$.

For the case where $d \geq 2$, we only have upper bounds on the approximation errors for $p = 2$ and $p = \infty$. The result for the former is stated in Theorem 2 from [3] while the latter is given in Theorem 3 from [13].

Theorem 2 *For any $f \in W^{r,2}(\Omega)$ and $\Sigma_n = \Gamma^{n,1}(ReLU; d, 1)$, we have*

$$E_n(f, \Sigma)_2 \leq C n^{-s/d} \|f\|_{W^{r,2}(\Omega)},$$

where $C = C(d)$.

Theorem 3 *For $K = U(Lip\,1)$ in the norm of $C(\Omega)$, we can calculate an upper bound in approximation for networks $\Sigma_n = \Gamma^{n,1}(ReLU; d, 1)$:*

$$E_n(K, \Sigma)_{C(\Omega)} \leq C \frac{\log_2 n}{n^{1/d}}$$

for $C = C(d)$.

Note that the upper bound in Theorem 2 depends on the target function itself, whereas Theorem 3 was able to provide an upper bound that only depends on the architecture of the network and the input dimension.

But single layer networks are not very capable. For instance it has been found that single layer (and even two layer) networks are not able to approximate all continuous multivariate functions [14] [15]. Many of the shortcomings of shallow neural networks are lifted with deeper architectures, and–for that reason–deep neural networks have been much more widely used in applications.

4.2.2 Deep ReLU Networks

While the concept of deep neural networks has been around since the inception of neural networks themselves, theoretical advancements in understanding their universal approximation properties as the depth increases only came about relatively recently.

First, we will state a result for a fixed width and arbitrary depth published by Zhou Lu *et al.* in 2017 [16]. The main result of their work [16] is given in Theorem 4

Theorem 4 *For any Lebesgue-integrable function $g : \mathbb{R}^m \to \mathbb{R}$, there exists a neural network γ with a ReLU activation, an arbitrary depth, and a width less than $m + 4$ at each hidden layer*

$$\gamma \in \bigcup_{j=1}^{\infty} \Gamma^{m+4,j}(ReLU; m, 1)$$

such that

$$\int_{\mathbb{R}^m} |\gamma(x) - g(x)|\mu(dx) < \epsilon$$

for any $\epsilon > 0$.

That is, Lu *et al.*'s result shows that the set of arbitrary depth, fixed width ReLU networks are dense in the set of Lebesgue-integrable functions. Theorem 4 is limited for two reasons. The first reason is that we have no bound on the depth of the neural network, meaning its applications are quite limited. Also, Theorem 4 is only proven for $d = 1$.

If we want to want to have depth bounds and also consider cases where $d \geq 2$, we can prove Theorem 5 as found in [3].

Theorem 5 *Let $s > \frac{d}{\tau} - \frac{d}{p} > 0$ for $\tau \leq \infty$ and $1 \leq p \leq \infty$, and $\Omega = [0, 1]^d$. Suppose $K = U(B_q^s(L_\tau(\Omega)))$ with $0 < q$, then*

$$E(K, \Sigma_{n[\log_2 n]^\beta})_{L_p(\Omega)} \leq C(s, d, \delta)n^{-s/d}$$

for $n \geq 1$, and the neural networks we are considering are of the form $\Sigma_n = \Gamma^{6d,Cn}(ReLU; d, 1)$, for $n \geq 1$, $C = C(s, d, \delta)$, $\beta = \max\{1, 2d/(s - \delta)\}$, and $\delta = s - \frac{d}{\tau} + \frac{d}{p}$.

If we modify our restrictions to functions in $U(\text{Lip } \alpha), 0 < \alpha \leq 1$, and $\Omega = [0, 1]^d$, deep ReLU networks have very fast rates of approximation as found by [17]. The univariate case is shown in Theorem 6.

Theorem 6 *Let $K = U(Lip\ 1)$ on $\Omega = [0, 1]$ and $\Sigma_n = \Gamma^{11,16n+2}(ReLU;1,1)$ for $n \geq 1$, then we have*

$$E(K, \Sigma_n)_{C(\Omega)} \leq 6n^{-2}.$$

And in 2020, Lu *et al.* [18] were able to improve Theorem 6 to provide upper error bounds for functions in $C^s([0, 1]^d)$ where $s \in \mathbb{N}$ as shown in Theorem 7.

Theorem 7 *For an $f \in C^s([0, 1]^d)$ where $s \in \mathbb{N}$, there exists a ReLU network $\gamma \in \Gamma^{W,L}(ReLU; d, 1)$ such that*

$$\|f - \gamma\|_{C([0,1]^d)} \leq c_1(s, d)\|f\|_{C^s([0,1]^d)}N^{-2s/d}L^{-2s/d}$$

for any $N, D \in \mathbb{N}$ and $W = c_2(s, d) \log_2(8N)$, $L = c_2(s)(D + 2) \log_2(4D) + 2d$.

In the case that $s \geq d$, Theorem 7 has an upper bound that scales at least as well as $O(N^{-2}L^{-2})$, which is comparable to the upper bound given by Theorem 6 (but this is counteracted by a large s^d factor in $c_1(s,d)$).

The $O(n^{-2})$ rate in Theorem 6 is very fast compared to standard approximations with only n parameters, which usually only have a rate of $O(n^{-1})$ [3]. This implies that the space of ReLU networks has space filling properties in $C(\Omega)$. While this seems like an excellent property due to the relatively simple formulation of ReLU networks, we need to keep in mind that the errors are given by the global best networks, and it might be numerically difficult to find such parameters which outpace standard approximation methods.

4.2.3 Deep v.s. Shallow Networks

When comparing deep and shallow networks, nonlinear functions seem to play an important role in distinguishing the capabilities between the two types of networks. In order to understand how deep networks are able to perform better on approximating nonlinear functions compared to shallow networks, we will need the following theorems [17].

Theorem 8 *Let $K = U(W^{r,\infty}([0,1]^d))$ for any $d, r \in \mathbb{N}$ and $\epsilon \in (0,1)$. Then, there exists a γ corresponding to a ReLU network with N weights and a depth L such that*

$$\|f - \gamma\|_{L^\infty([0,1]^d)} < \epsilon, \quad f \in K,$$

where $N \leq c\epsilon^{-d/r}(\ln(1/\epsilon) + 1)$ and $L \leq c(\ln(1/\epsilon) + 1)$ for some constant $c = c(d,r)$.

Theorem 9 *Let $f \in C^2([0,1]^d)$ be a nonlinear function. Then, for any fixed L, if a ReLU network γ with N weights and a depth L approximates f to within some*

$$\|f - \gamma\|_{C^2([0,1]^d)} < \epsilon,$$

then it must be that $N \geq c\epsilon^{-1/(2(L-2))}$.

Theorem 8 provides an upper bound for the total number of weights in the model, which increases at a rate of $O(\epsilon^{-d/r}\ln(1/\epsilon))$; and, for a fixed depth L, Theorem 2 provides a lower bound which increases at a rate of $O(\epsilon^{-1/(2(L-2))})$. This means that when

$$\frac{d}{n} < \frac{1}{2(L-2)},$$

and as ϵ tends to 0, the number of weights required by deep networks to achieve an error ϵ will be lower than that of the shallow network. Additionally, this effect will be exaggerated for smoother nonlinear functions.

So, in a sense, deep ReLU networks are very efficient approximators while shallow ReLU networks can be very poor approximators in comparison.

4.2.4 Adaptive Width Networks

So far, all previous approximation theorems we explored tried to find the best possible set of parameters for a fixed architecture in order to approximate a function. However, if we allow for the architecture to be 'adaptive', then we could expect some performance gains. Indeed, Yarotsky [17] was able to prove such a result with Theorem 10.

Theorem 10 *For any $f \in U(W^{1,\infty}([0,1]))$ and $\epsilon \in (0, 1/2)$, there exists a ReLU network γ with depth 6 such that*

$$\|f - \gamma\|_{L^\infty([0,1])} < \epsilon,$$

while having less than $\frac{c}{\epsilon \ln(1/\epsilon)}$ total weights, where c is an absolute constant.

Note that in Theorem 10, we allow for γ to be 'tailored' to each chosen function f, rather than specifying an architecture to approximate a whole class of functions with an error better than ϵ.

4.3 The Curse Of Dimensionality

It is worth mentioning how neural networks fare against the curse of dimensionality, since we have discussed the ability of neural networks to approximate functions with multi-dimensional inputs.

The curse of dimensionality is best described with an example. Say we have a 3-dimensional unit cube. Then, to cover 20% of the cube, we need to cover a volume of 0.2, which corresponds to a cube with a side length of $\sqrt[3]{0.2} \approx 0.58$. If we scale this up to a $d = 20$ cube, then to explore 20% of that cube, we would need another cube with a side length of $\sqrt[20]{0.2} \approx 0.92$. In general, as $d \gg 1$, the size of the cube we would need to cover the same proportion of the volume grows very quickly.

This is an issue for us because a naive way to approximate functions to a precision ϵ on $\Omega = [0,1]^d$ would be to discretize the domain to small cubes at localize the approximation for the neural network at each cube. So for example [19], it would not be unusual to see a statement like Theorem 11.

Theorem 11 *Let $f : [0,1]^d \to \mathbb{R}$ be in Lip 1 and $\epsilon > 0$. Then, there exists a ReLU network $\gamma \in \Gamma^{W,3}$ such that*

$$\|f - \gamma\|_{L^1} \leq \epsilon,$$

where W is on the order of $O(1/\epsilon^d)$.

But the scaling given by Theorem 11 is clearly undesirable due to its $1/\epsilon^d$ scaling, especially since neural networks are frequently used in high-dimensional contexts such as image recognition.

However, Barron's Theorem [20] shows that neural networks are able to overcome this hurdle. Barron's theorem relies on sigmoidal activation functions as defined in Definition 13, and uses those to construct a a three-layer network which scales favorably.

Definition 13 *A continuous function $f : \mathbb{R} \to \mathbb{R}$ is sigmoidal if it satisfies*

$$\lim_{x \to \infty} f(x) = 1$$

$$\lim_{x \to -\infty} f(x) = 0$$

Theorem 12 *For $f : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^m$, and $\epsilon > 0$, there exists a two-layer neural network γ with a sigmoidal activation of width less than*

$$k \leq \frac{8\, Vol(\Omega)}{\epsilon^2}(8\pi C)^2$$

such that

$$\|f - \gamma\|_2 \leq \epsilon,$$

where

$$C = \int_{\mathbb{R}^m} \|w\|_2 |\hat{f}(w)| dw$$

is required to be finite, and

$$\hat{f}(w) = \int_{\Omega} e^{-2\pi i w \cdot x} f(x) dx.$$

For the case where $\Omega = [0, 1]^d$ Barron's Theorem, as stated in 12 shows that we can maintain a parameter scaling of $O(\frac{C^2}{\epsilon^2})$ which is much better than $O(1/\epsilon^d)$ found in Theorem 11. So, neural networks are somewhat able to avoid the curse of dimensionality.

5 Transformers

Neural networks have been wildly successful at solving many different types of problems, and new types of models which have been tailored to specific tasks to gain an advantage over simple artificial neural networks are constantly being explored [21].

One of the most recently developed models that has made a significant impact in machine learning research is the 'transformer'. Transformer models, while originally developed in 2017 for the purpose of text translation [12], have been utilized in a wide range of generative AI applications from text and image generation [12] [22] [23] to time series prediction [24].

At the heart of all these transformer models is a specific step in the computation known as a 'transformer block'. Transformer blocks, somewhat akin to layers in an artificial neural network, are composed with each other and then embedded as a part of a larger model that combines neural networks as well as possibly other architectures to produce an output; and transformer blocks themselves are made up of activation normalization steps, a feed-forward network, and a series of linear transformations plus biases known as 'self-attention' layers which is the core mechanism of a transformer block.

5.1 The Transformer Network

Most modern implementations of transformer models in generative AI applications are known as decoder-only, generative transformers (from here on, any mention of 'transformer' will be referring to decoder-only, generative transformers). We will now walk through a basic transformer model from the input to the output, step by step, covering all the necessary computations throughout the model.

5.1.1 Inputs and Outputs of Transformers

Decoder-only generative transformers usually take a sequence of d_{model} dimensional vectors as the input and then output a singular vector of dimension d. While transformer models can handle a variable sequence length, they are usually only trained on a maximum sequence length: meaning that it is possible to use sequences shorter than the maximum sequence length, but they will not be able to handle sequences longer than that maximum very well. The maximum number of vectors in the sequence that the model can handle is known as the context window of the transformer.

In natural language settings, the goal is usually to be to predict the next token given a sequence of tokens. A token is a small string–usually only a few characters–that is assigned a unique id, and all the tokens that we assign an id to make up the vocabulary of the model. For more information on how tokenization algorithms break apart sentences, refer to [25] [26] and [27], which are all popular algorithms to tokenize text. The tokenized text is then inputted into an embedding model which prescribes quantifiable meaning to the tokens. Embedding algorithms can be as basic as assigning a vector to each token [28] or they can take contextual clues into account [29]. The tokenizer and the embedding method, together, are able to convert a string of text into a sequence of vectors–with one vector of dimension d for each token.

The output of the transformer in natural language settings is still a d_{model} dimensional vector, but in order to predict the next token, the vector is then used to assign a probability among all the tokens in the vocabulary. One common way to do this is to apply a decoder matrix $W_{dec} \in \mathbb{R}^{d_{\text{vocab}} \times d_{\text{model}}}$ to the output vector $u \in \mathbb{R}^{d_{\text{model}}}$, where d_{vocab} is the size of the vocabulary, and then apply a softmax operation to the vector to obtain a probability for each token.

5.1.2 Self-Attention

For a given sequence of T vectors, $x_{[T]} \in \mathbb{R}^{d_{\text{model}}}$, we can arrange the sequence into an array $X \in \mathbb{R}^{T \times d_{\text{model}}}$:

$$X = [x_1, x_2, \ldots, x_T].$$

Then, we compute $Q, K, V \in \mathbb{R}^{T \times d}$, the queries, keys, and values, respectively, as

$$Q(X) = XW^Q$$

$$K(X) = XW^K$$
$$V(X) = XW^V,$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d}$ are the learnable hyperparameters of the model, also referred to as the query, key, and value matrices/weights. For an explanation of the names of the query, key, and value matrices, refer to [12] [30]. Then, attention [12] is computed as

$$\text{attn}(Q, K, V) = \mathcal{S}\left(\text{MASK}\left(\frac{QK^\top}{\sqrt{d}}\right)\right)V. \tag{8}$$

In Equation (8), we introduced three important features of transformers: the scaled dot product, the causal mask, MASK, and the softmax operation $\mathcal{S}$.

The scaling of the dot product, $QK^\top$, by a factor of $\sqrt{d}$ serves as a way to normalize the values of the $QK^\top$ as d_{model} increases. For example, if we assume that each row in Q and K have elements with mean 0 and variance 1, then the dot products of each row in Q with those in K will be of the form $\sum_{i=1}^{d} q_i k_i$, and will thus have mean 0 and variance d [12].

The causal mask is a simple operation that sets certain elements in a matrix to $-\infty$:

$$\text{MASK}(M)_{ij} = \begin{cases} M_{ij}, & i \geq j \\ -\infty, & i < j \end{cases}.$$

The effect of applying a causal map to the scaled dot product is that the queries earlier in the sequence will not be able to 'see' keys after its position in the sequence. Otherwise, we would have data leakage since the model will be learning to predict the next token based on previous tokens as well as future tokens. The mask ensures that the softmax will ignore the effects of future tokens. The causal map is unique to generative transformers and is also only necessary during the training phase, since during inference we only need to predict one token at a time, so only the last row of Q is used (whereas during training, it is usually more computationally efficient to compute multiple next tokens with one matrix multiplication, which we will achieve with $QK^\top$).

The matrix $\text{MASK}\left(\frac{QK^\top}{\sqrt{d}}\right) \in \mathbb{R}^{T \times T}$ gives us some sort of 'attention score' for each pair of tokens according to W^Q and W^V. The higher the score relative to other pairs, the more important the connection between those two tokens is.

The softmax operation, $\mathcal{S} : \mathbb{R}^{T \times T} \to \mathbb{R}^{T \times T}$, is applied to $\text{MASK}\left(\frac{QK^\top}{\sqrt{d}}\right)$ in a row-wise fashion. For an array $A \in \mathbb{R}^{m \times n}$, we have the i, j element of $\mathcal{S}(A)$ being equal to

$$\mathcal{S}_{ij}(A) = \frac{\exp(A_{ij})}{\sum_{k=1}^{n} \exp(A_{ik})}. \tag{9}$$

Then, multiplying $\mathcal{S}\left(\text{MASK}\left(\frac{QK^\top}{\sqrt{d}}\right)\right)$ with V gives us a type of weighted average among the values of each token (the rows of V).

5.1.3 Multi-Head Attention

Multi-head attention (MHA) is a simple extension of self-attention in that we simply sum over multiple attn functions with a different set of W^Q, W^K, W^V each time. Then, we use another linear transformation W^O to project each attn output back to d_{model}.

Concretely, MHA is calculated as

$$\text{MHA}(X) = \sum_{i=1}^{h} \text{attn}(XW_i^Q, XW_i^K, XW_i^V)W_i^O. \tag{10}$$

In Equation 10, each i index denotes a different 'head' where we have a new set of $W_i^Q, W_i^K, W_i^V,$ and W_i^O. To remind ourselves of the shapes of the hyperparameters, we have $W_{[h]}^Q, W_{[h]}^K, W_{[h]}^V \in \mathbb{R}^{d_{\text{model}} \times d}$ and $W_{[h]}^O \in \mathbb{R}^{d \times d_{\text{model}}}$.

5.1.4 Positional Embeddings

If you ignore the mask, the attn function in Equation (8) has the unique property that you can permute the rows of rows of X to obtain Q', K, V', and you will have that $\text{attn}(Q', K', V')$ is simply $\text{attn}(Q, K, V)$ permuted in an identical way. This is called permutational equivariance.

Definition 14 *A function $f : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$, for any $m, n \in \mathbb{N}$, is permutation equivariant if for any permutation matrix P, f satisfies*

$$f(PX) = Pf(X), \quad \forall\, X \in \mathbb{R}^{m \times n}.$$

Intuitively, this means that you can swap any two rows of X and the result of applying f to the permuted X will be equivalent to permuting the rows of $f(X)$ in the same way.

The proof for the permutational equivariance of self-attention without any masking step (and the masking step is usually only done during training) can be found in Section A.

The consequences of this in language models is that the transformer cannot learn any differences between different orderings of tokens for sequences of the same length. To prescribe meaning to different orders, we will need to use positional embeddings. Positional embeddings are added to the token embeddings to produce an array that has both information about the meaning of individual tokens as well as the meaning of the position in the context window.

While there are multiple ways to implement positional embeddings [31], the general idea is that for some sequence of token embeddings X, we obtain a new encoding, E, which takes positional information into account with a linear transformation on X plus X:

$$E = X + P_e X,$$

where P_e is the linear transformation on X.

Positional embeddings are not crucial to understanding the mechanisms of transformers, so they are left out in this walkthrough of transformer models inconsequentially, but we will revisit them.

5.1.5 Layer Normalization

In each transformer block, we apply layer normalization steps multiple times. Currently, there are two popular layer normalization techniques: LayerNorm [32] and root means square layer normalization (RMSNorm) [33].

Both LayerNorm and RMSNorm are functions of activations, which in our case, will be the elements of an array of dimension $\mathbb{R}^{T \times d_{\text{model}}}$.

For either normalization method, we first compute the mean and standard deviation of the activations for the rows of an array $X \in \mathbb{R}^{T \times d_{\text{model}}}$:

$$\mu^l(X)_i = \frac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} X_{ij}$$

$$\sigma^l(X)_i = \sqrt{\frac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} \left(X_{ij} - \mu^l(X)_i\right)^2}.$$

Then, the elements of the LayerNorm-ed array are computed as

$$\text{LayerNorm}(X)_{ij} = \frac{X_{ij} - \mu^l(X)_i}{\sigma^l(X)_i}.$$

RMSNorm, on the other hand, does not have the $(-\mu^l(X)\mathbf{1}_{T \times d_{\text{model}}})$ term:

$$\text{RMSNorm}(X)_{ij} = \frac{X_{ij}}{\sigma^l(X)_i}.$$

The $(-\mu^l(X)\mathbf{1}_{T \times d_{\text{model}}})$ term serves to center the activations, but destroys information in the process, disallowing a perfect reconstruction of the original activations from the LayerNorm's output [34]. While some hypothesize that RMSNorm performs better than LayerNorm due to the fact it allows us to fully reconstruct the original activations from its output, multiple state-of-the-art models have been built with LayerNorm [35] [36] as well as RMSNorm [37] [22].

5.1.6 Transformer Blocks

Now, we can build the transformer block, the basic units of transformer networks. There are many different implementations of transformer blocks, but essentially, almost all of them are sums of compositions of attention layers, layer normalizations, and feed forward networks.

As an example, say we have an input $X \in \mathbb{R}^{T \times d_{\text{model}}}$, then a reasonable transformer block would be given by $T_i : \mathbb{R}^{T \times d_{\text{model}}} \to \mathbb{R}^{T \times d_{\text{model}}}$

$$\tilde{T}_i(X) = \text{LayerNorm}(X + \text{MHA}_i(X)) \tag{11}$$

$$T_i(X) = \text{LayerNorm}(f_{\theta_i}(\tilde{T}_i(X)) + \tilde{T}_i(X)), \tag{12}$$

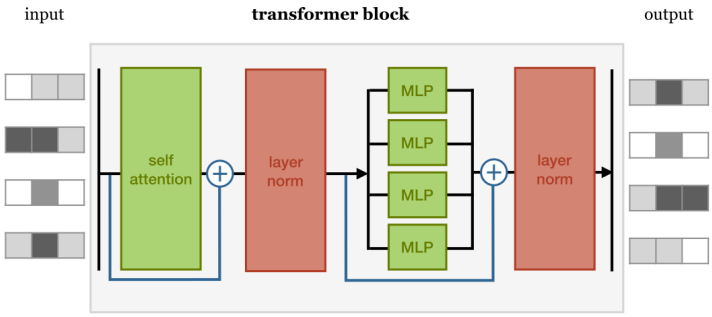

Figure 3: A standard architecture of a transformer block which has a self attention layer, a layer normalization step, a feedforward network, and another layer-normalization step. The + symbols represent residual connections [38]. Image courtesy of Peter Bloem [30].

where $f_{\theta_i} : \mathbb{R}^{T \times d_{\text{model}}} \to \mathbb{R}^{T \times d_{\text{model}}}$ is a feed-forward network (usually consisting of only one layer) with parameters θ_i where the i index indicates that we can (and usually do) have multiple transformer blocks in a single transformer network, each with their own parameters of query, key, and value weights for the multi-head attention and θ_i for the feed-forward network.

You may notice that in Equation (11) we add X to its multi-head attention output, and in Equation (12) we add $\tilde{T}_i(X)$ to its feed-forward network output. Such operations are known as residual connections [38]. Without going too far into the details, residual connections were originally intended to solve the vanishing gradient problem [39], which is when gradient updates to parameters hardly affect the parameters in the first few layers when compared to parameters in the last few layers–decreasing the effective width of the network. Transformer networks, as we will see shortly, can also become very deep, so residual connections become very useful to be able to update parameters throughout the network.

Instead of the explicit form of T_i in Equation (12), it is often more helpful to visualize the function in a diagram as given by Figure 3 which communicate the basic sequence of the Transformer block while leaving smaller details such as the exact architecture of the feed-forward networks up to the reader to fill in.

5.1.7 Transformer Model

With a well-defined transformer block, we can build a full transformer network. A very basic transformer network, $\text{TN}(\cdot) : \mathbb{R}^{T \times d_{\text{model}}} \to \mathbb{R}^{T \times d_{\text{model}}}$, can simply by a composition of $b \in \mathbb{N}$ transformer blocks

$$\text{TN}(X) = T_b \circ T_{b-1} \circ \ldots \circ T_1(E(X)), \tag{13}$$

where each transformer block T_i is also a mapping from $\mathbb{R}^{T \times d_{\text{model}}}$ to $\mathbb{R}^{T \times d_{\text{model}}}$, and $E(X)$ applies the positional embeddings onto X. Again, a visualization of an example network is provided in 4.

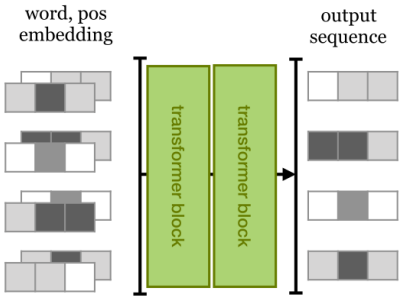

Figure 4: An example transformer network comprised of two transformer blocks. Most implementations of large language models can have 50+ transformer blocks, but the overall structure remains very similar [37].

As a concrete example, for text generative transformers, X is a sequence of T embeddings each of dimension d_{model}, and $E(X)$ gives us the positionally embedded sequence, which gets passed into a series of transformer blocks and outputs another sequence of vectors with a similar length and dimension. Typically, each vector in the output sequence will also have an additional linear transform which 'unembeds' the vector to obtain a probability distribution over all tokens in the vocabulary, which serves to determine the most likely tokens to follow the given sequence.

5.2 Universal Approximation Theorems of Transformers

In this section, we will be exploring the classes of functions that transformers are able to approximate. We will explore transformers as universal approximators of sequence-to-sequence functions by studying work done by Yun *et al.* [40].

Definition 15 *Sequence-to-sequence functions map a sequence of some fixed number of $M \in \mathbb{N}$ vectors of dimension m to another sequence of $N \in \mathbb{N}$ vectors of dimension n. These sequences can be arranged into matrices in $\mathbb{R}^{M\times m}$ for the inputs and matrices in $\mathbb{R}^{N\times n}$ for the outputs. That is, we can express any sequence to sequence function, f as*

$$f : \mathbb{R}^{M\times m} \to \mathbb{R}^{N\times n}.$$

Some sequence-to-sequence functions can have the property of being permutationally equivariant as defined in Definition 14.

We will use $\mathcal{F}_{\text{PE}}(\mathbb{R}^{m\times n})$ to denote the class of all continuous permutationally equivariant functions with compact support that map $\mathbb{R}^{m\times n}$ to $\mathbb{R}^{m\times n}$ for some choice of $m, n \in \mathbb{N}$. Continuity in $\mathcal{F}_{\text{PE}}$ will be enforced with an entry-wise ℓ^p norm for $1 \leq p < \infty$. That is, a function $f \in \mathcal{F}_{\text{PE}}(\mathbb{R}^{m\times n})$ is continuous at some

sequence X_0 under the entry-wise ℓ^p norm if for any $\epsilon > 0$, there exists a $\delta > 0$ such that

$$\left(\sum_{i \in [m], j \in [n]} |(X_0)_{ij} - X_{ij}|^p \right)^{1/p} < \delta \implies \left(\sum_{i \in [m], j \in [n]} |f(X_0)_{ij} - f(X)_{ij}|^p \right)^{1/p} < \epsilon.$$

The metric used define a distance between two functions $f_1, f_2 \in \mathcal{F}_{\text{PE}}(\mathbb{R}^{m \times n})$ is similarly

$$d_p(f_1, f_2) = \left(\int \|f_1(X) - f_2(X)\|_p^p dX \right)^{1/p}.$$

To study the sequence-to-sequence approximation properties of transformers, the form of self-attention we will be studying is given by

$$\text{attn}(X) = X + \sum_{i=1}^{h} \mathcal{S}((XW_i^Q)(XW_i^K)^\top) XW_i^V W_i^O, \tag{14}$$

where all the terms and hyperparameters in the equation are similar to their corresponding terms in Equations (8) and (10). If we take $X \in \mathbb{R}^{T \times d_{\text{model}}}$, the shapes of the hyperparameters are $W_{[h]}^Q, W_{[h]}^K, W_{[h]}^V \in \mathbb{R}^{d_{\text{model}} \times d}, W_{[h]}^O \in \mathbb{R}^{d \times d_{\text{model}}}$. However, note that this is a redefinition from Equation (8) since we no longer use a mask.

5.2.1 No Positional Embeddings

Without an positional embeddings, the transformer block is given by

$$t^{h,m,r}(X) = \text{attn}(X) + \text{ReLU}(\text{attn}(X)W_1 + \mathbf{1}_T b_1^\top)W_2 + \mathbf{1}_T b_2^\top, \tag{15}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}} \times r}, W_2 \in \mathbb{R}^{r \times d_{\text{model}}}, b_1 \in \mathbb{R}^r, b_2 \in \mathbb{R}^{d_{\text{model}}}$ and we use the attn function defined in Equation (14). In Equation (14), h represents the number of heads and d represents the head size; and in Equation (15), the parameter r represents the size of the hidden layer of the feed forward network in Equation (12).

With a solid definition of a transformer block, we can now define our set of transformer networks, $\mathcal{T}^{h,d,r}$, as the composition of multiple transformer blocks with h heads of dimension d and a feed forward network with a hidden layer of dimension r:

$$\mathcal{T}^{h,m,r} = \{g : \mathbb{R}^{T \times d_{\text{model}}} \to \mathbb{R}^{T \times d_{\text{model}}} | g \text{ is a composition of blocks } t^{h,m,r}\}.$$

For any $g \in \mathcal{T}^{h,m,r}$, the composed transformer blocks do not necessarily share the same query, key, and value weights.

Note that the only difference between the transformer block given by Equation (15) and the transformer block defined in Equation (13) is that we don't have a layer normalization step and our attention function is slightly modified. Still, we maintain many important features of transformer blocks such as permutational equivariance:

Theorem 13 *The transformer block given by $t^{h,m,r}$ is permutation equivariant. That is,*

$$t^{h,m,r}(PX) = Pt^{h,m,r}(X)$$

for any permutation matrix P.

The proof for Theorem 13 will be given in Section A.

The fact that transformer blocks are permutation equivariant means that any transformer network in $\mathcal{T}^{h,m,r}$ will also be permutation equivariant, so it is natural to assume that they will be able to approximate some subset of the class of all permutationally equivariant functions. Indeed, the main result by Yun *et al.* shown in Theorem 14 found that transformer networks are able to approximate all continuous, permutationally equivariant functions with compact support arbitrarily well.

Theorem 14 *Let $1 < p < \infty$ and $\epsilon > 0$, then for any $f \in \mathcal{F}_{PE}(\mathbb{R}^{T \times d_{model}})$, there exists a transformer network $g \in \mathcal{T}^{2,1,4}$, such that $d_p(f, g) \leq \epsilon$ on the set Ω that f has compact support on.*

5.2.2 Using Positional Embeddings

We can break the permutation equivariance property of transformer networks with positional embeddings. We can incorporate positional embeddings into our transformer networks by simply adding some embedding $E \in \mathbb{R}^{T \times d_{\text{model}}}$ to the original input X. Then, let

$$\mathcal{T}_P^{h,m,r} = \{g_P(X) = g(X + E) | g \in \mathcal{T}^{h,m,r}, E \in \mathbb{R}^{T \times d_{\text{model}}}\}$$

be the class of all transformer networks with positional embeddings.

By studying the approximation properties of transformer networks with positional embeddings, Yun *et al.* was able to obtain Theorem 15 which states that transformer networks can approximate any function in $\mathcal{F}_{\text{CD}}$, the set of all functions that are continuous on a compact set and map $\mathbb{R}^{T \times d_{\text{model}}}$ to $\mathbb{R}^{T \times d_{\text{model}}}$.

Theorem 15 *Let $1 < p < \infty$ and $\epsilon > 0$, then for any $f \in \mathcal{F}_{CD}(\mathbb{R}^{T \times d_{model}})$, there exists a transformer network $g \in \mathcal{T}_P^{2,1,4}$, such that $d_p(f, g) \leq \epsilon$ on the set Ω that f has compact support on.*

Together, Theorems 14 and 15 tell us about the approximation properties of bounded width, arbitrary depth transformer networks (in the sense that we specify the head size, head dimension, and hidden layer width, but allow for an arbitrary amount of transformer blocks to be composed together.) What is especially surprising about the result of these theorems is that the parameters (h, m, r) needed to approximate $\mathcal{F}_{\text{CD}}$ or $\mathcal{F}_{\text{PE}}$ do not have a dependence on T or d_{model}. Also, despite self-attention only being able to capture pair-wise interactions, it is interesting to see that the class of functions that transformers can approximate well is reasonably rich.

6 Discussion

Neural networks have been very useful in a wide range of applications, and recently theoretical guarantees for their approximation properties in Section 4.2 have given us insight into why neural networks are so effective in approximating continuous functions.

Transformers networks as defined in Section 5 are among some of the latest iterations of a deep learning architecture, and they, too, have been extremely effective at solving lots of tasks. Due to their versatility, we looked into their universal approximation properties in Section 5.2 and found that they are able to approximate sequence to sequence functions with permutational equivariance, the permutational equivariance condition is relaxed when we allow for positional embeddings. Despite the fact that we only have results for the bounded width and arbitrary depth case, we were able to observe how transformer networks only need relatively small architectures to well-approximate sequence-to-sequence functions.

Our discussion has provided insight into the types of functions neural networks and transformer networks are able to approximate arbitrarily well. As transformers are used in larger and more capable networks, understanding their universal approximating properties will be very useful in making sure that transformer networks scale efficiently, and so more research into the classes of functions they are able approximate is needed, especially in the regimes of bounded width and bounded depth.

References

[1] T. Arbogast and J. L. Bona, "Methods of Applied Mathematics."

[2] H. Brezis, *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer Science & Business Media, Nov. 2010. Google-Books-ID: GAA2XqOIIGoC.

[3] R. DeVore, B. Hanin, and G. Petrova, "Neural network approximation," *Acta Numerica*, vol. 30, pp. 327–444, May 2021.

[4] S. C. Peter, J. K. Dhanjal, V. Malik, N. Radhakrishnan, M. Jayakanthan, and D. Sundar, "Quantitative Structure-Activity Relationship (QSAR): Modeling Approaches to Biological Applications," in *Encyclopedia of Bioinformatics and Computational Biology* (S. Ranganathan, M. Gribskov, K. Nakai, and C. Schnbach, eds.), pp. 661–676, Oxford: Academic Press, Jan. 2019.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, Aug. 2006. Google-Books-ID: qWPwnQEACAAJ.

[6] R. A. Alzahrani and A. C. Parker, "Neuromorphic circuits with neural modulation enhancing the information content of neural signaling," in *In-*

ternational Conference on Neuromorphic Systems 2020, ICONS 2020, (New York, NY, USA), Association for Computing Machinery, 2020.

[7] W. Commons, "File:artificial neuron structure.svg — wikimedia commons, the free media repository," 2024. [Online; accessed 22-April-2025].

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[9] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015. Publisher: Nature Publishing Group.

[11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[13] F. Bach, "Breaking the curse of dimensionality with convex neural networks," *Journal of Machine Learning Research*, vol. 18, no. 19, pp. 1–53, 2017.

[14] N. J. Guliyev and V. E. Ismailov, "On the approximation by single hidden layer feedforward neural networks with fixed weights," *Neural Networks*, vol. 98, pp. 296–304, Feb. 2018.

[15] N. J. Guliyev and V. E. Ismailov, "Approximation capability of two hidden layer feedforward neural networks with fixed weights," *Neurocomputing*, vol. 316, pp. 262–269, 2018.

[16] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," *Advances in neural information processing systems*, vol. 30, 2017.

[17] D. Yarotsky, "Error bounds for approximations with deep ReLU networks," *Neural Networks*, vol. 94, pp. 103–114, Oct. 2017.

[18] J. Lu, Z. Shen, H. Yang, and S. Zhang, "Deep network approximation for smooth functions," *SIAM Journal on Mathematical Analysis*, vol. 53, no. 5, pp. 5465–5506, 2021.

[19] H. Hadiji, "Theoretical Principles of Deep Learning - Class II: Approximation with Neural nets," December 2024.

[20] A. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[21] J. Schneider and M. Vlachos, "A survey of deep learning: From activations to transformers," in *ICAART (2)*, pp. 419–430, 2024.

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozire, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," Feb. 2023. arXiv:2302.13971 [cs].

[23] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182, 2023.

[24] A. F. Ansari, L. Stella, A. C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor, J. Zschiegner, D. C. Maddix, H. Wang, M. W. Mahoney, K. Torkkola, A. G. Wilson, M. Bohlke-Schneider, and B. Wang, "Chronos: Learning the language of time series," *Transactions on Machine Learning Research*, 2024. Expert Certification.

[25] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (K. Erk and N. A. Smith, eds.), (Berlin, Germany), pp. 1715–1725, Association for Computational Linguistics, Aug. 2016.

[26] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152, Mar. 2012. ISSN: 2379-190X.

[27] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (E. Blanco and W. Lu, eds.), (Brussels, Belgium), pp. 66–71, Association for Computational Linguistics, Nov. 2018.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[29] P. Dufter, M. Schmitt, and H. Schtze, "Position information in transformers: An overview," *Computational Linguistics*, vol. 48, pp. 733–763, 09 2022.

[30] P. Bloem, "Transformers from scratch," Aug. 2019.

[31] L. Zhao, X. Feng, X. Feng, W. Zhong, D. Xu, Q. Yang, H. Liu, B. Qin, and T. Liu, "Length extrapolation of transformers: A survey from the perspective of positional encoding," *arXiv preprint arXiv:2312.17044*, 2023.

[32] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," July 2016. arXiv:1607.06450 [stat].

[33] B. Zhang and R. Sennrich, "Root mean square layer normalization," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[34] A. Gupta, A. Ozdemir, and G. Anumanchipalli, "Geometric Interpretation of Layer Normalization and a Comparative Analysis with RMSNorm," Feb. 2025. arXiv:2409.12951 [cs].

[35] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," *Unpublished Work*, June 2018.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[37] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou,

Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, "DeepSeek-V3 Technical Report," Feb. 2025. arXiv:2412.19437 [cs].

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[39] J. F. Kolen and S. C. Kremer, "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies," in *A Field Guide to Dynamical Recurrent Networks*, pp. 237–243, IEEE, 2001.

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

A Permutational Equivariance of Transformer Blocks

To prove that transformer blocks (15) are permutationally equivariant, we need to show that

$$t^{h,m,r}(PX) = Pt^{h,m,r}(X).$$

This will be done in two steps. First, we will show that the attn function (14) is permutationally equivariant. Suppose we have an input and any permutation matrix $X, P \in \mathbb{R}^{T\times d_{\text{model}}}$, then note that for a permuted input PX,

$$(PXW^Q)(PXW^K)^\top = P(XW^Q)(XW^K)^\top P^\top.$$

Since the softmax operation is row-wise, it then follows that

$$\begin{aligned}
\text{attn}(Q(PX), K(PX), V(PX)) &= PX + \sum_{i=1}^{h} P\mathcal{S}((XW_i^Q)(XW_i^K)^\top)P^\top PXW_i^V W_i^O \\
&= PX + \sum_{i=1}^{h} P\mathcal{S}((XW_i^Q)(XW_i^K)^\top)XW_i^V W_i^O \\
&= P\left(X + \sum_{i=1}^{h} \mathcal{S}((XW_i^Q)(XW_i^K)^\top)XW_i^V W_i^O\right) \\
&= P\ \text{attn}(Q(X), K(X), V(X)),
\end{aligned}$$

where we used the fact that $P^\top P = I$. Next, for the feed-forward layer, we have

$$\begin{aligned}
t^{h,m,r}(PX) &= P\ \text{attn}(X) + \text{ReLU}(P\ \text{attn}(X)W_1 + P\mathbf{1}_T b_1^\top)W_2 + P\mathbf{1}_T b_2^\top \\
&= P\ t^{h,m,r}(X)
\end{aligned}$$

where we used the fact that the ReLU activation function is permutationally equivariant since it acts element-wise on its input, and we permuted the biases by assuming that the feedforward network acts token-wise.