Transformers, Roll Out:
From Kernel Regression to Self-Attention

Rishabh Anand e0555795@u.nus.edu
A0220603Y

Ryan Chung Yi Sheng e0550360@u.nus.edu
A0219702J

Abstract

Transformers (Vaswani et al., 2017) have become the go-to model for a wide range of tasks in Machine Learning, especially in natural language processing. This project breaks down Transformers into its key parts, explaining why each component is designed the way it is. We make connections to the kernel machines literature, helping us understand the transformer's inner workings that drive its stellar performance. We aim to make the mathematics behind Transformers more accessible, helping readers grasp why they are so effective. Source code is available here.

1 Introduction

Project motivation. In the era of advanced large language models like ChatGPT, understanding their underlying architecture and geometric interpretation is crucial. In this project, we delve into the Transformer by Vaswani et al. (2017) – the backbone of ChatGPT – exploring how it enables learning on sequential data. We dive into the history behind the *self-attention mechanism*, the main driver of the transformer, by connecting it to humbler beginnings in the *kernel machines* literature. We first explore the explicit connections between kernels and attention before scaling the latter for sequence learning experiments.

Report structure. We first introduce kernel regression, specifically focusing on the Nadaraya-Watson kernel regression estimator, which is the predecessor of modern-day attention. We then formally introduce attention, as well as self-attention. Next, we introduce how self-attention can be stacked up to form the Transformer, a powerful sequence learning model, along with its details, and demonstrate its efficacy at learning periodic functions. We conclude with an analysis and discussion of the results.

Notation. Throughout this report, we use lowercase bolded characters ($\mathbf{a}, \mathbf{b}$) to represent vectors and uppercase bolded characters for matrices ($\mathbf{W}, \mathbf{X}$). Within a vector, the index of its elements along its length is represented by a bracketed superscript ($\mathbf{a}^{(i)}$ is the i-th element in the vector). Among a collection of vectors, a subscript represents its index within the collection ($\mathbf{a}_j$ refers to the j-th element of a set S).

2 Kernel Regression

We initially diverge from Transformers and delve into the predecessor of the attention mechanism, the Nadaraya-Watson kernel regression estimator (Smola & Zhang, 2019). Our objective is to revisit this traditional algorithm to deepen our comprehension of the attention mechanism.

2.1 Linear Smoother

We consider a set of independent and identically distributed (i.i.d.) samples denoted as $(\mathbf{x}_i,\ y_i)$ from the model:

$$y_i = m(\mathbf{x}_i) + \epsilon_i, \quad i = 1, \cdots, N \tag{1}$$

we try to estimate m using some function $\hat{m}$. If $\hat{m}$ comes from the class of functions with the form

$$\hat{m}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) \cdot y_i = \alpha(\mathbf{x})^T \mathbf{y} \tag{2}$$

then $\hat{m}$ is a linear smoother.

As an illustration, linear regression is a typical example of a linear smoother. The closed-form solution for $\hat{\mathbf{\Theta}}$ is given by:

$$\hat{\mathbf{\Theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Therefore we can express $\hat{m}$ in the form of Equation 2 like so:

$$\hat{m}(\mathbf{x}') = \mathbf{x}'^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \alpha(\mathbf{x}')^T\mathbf{y}$$

where $\alpha(\mathbf{x}')^T = \mathbf{x}'^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$

2.2 Kernel Density Estimation

We will provide a brief overview of Kernel Density Estimation (KDE), but note that the detailed explanations are beyond the scope of our project (Tibshirani, 2014). A one-dimensional smoothing kernel is any smooth function K such that it has the following properties:

1. Symmetry: $K(x) = K(-x)$
2. Normalization: $\int K(x)dx = 1$
3. Kernel of Order 1: $\int xK(x)dx = 0$
4. Positive Second Moment: $\int x^2K(x)dx > 0$

A few examples of kernels are:

1. Gaussian: $K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$
2. Boxcar: $K(x) = \frac{1}{2}$ if $|x| \leq 1$
3. Epanechikov: $K(x) = \frac{3}{4}(1 - x^2)$ if $|x| \leq 1$

We consider a set of independent and identically distributed (i.i.d.) samples denoted as x_i drawn from some univariate distribution with unknown density f. Then its kernel density estimator is

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} K_{h_1}(x - x_i) = \frac{1}{Nh_1}\sum_{i=1}^{N} K\Big(\frac{x - x_i}{h_1}\Big) \tag{3}$$

where $h_1 > 0$ is the bandwidth of the kernel function and $K_{h_1}(x) = \frac{1}{h_1}K(\frac{x}{h_1})$. We won't delve deeply into the specifics, but by utilizing a product kernel, we can conduct multivariate density estimation, leading to the following formula:

$$\hat{f}(x, y) = \frac{1}{N}\sum_{i=1}^{N} K_{h_1}(x - x_i)K_{h_2}(y - y_i) \tag{4}$$

2.3 Nadaraya-Watson Kernel Regression

With the necessary foundations laid, let's now shift our focus to another type of linear smoother: the Nadaraya-Watson kernel regression estimator. We will provide the derivation of this estimator. In the following sections, we'll draw upon the concepts introduced in this chapter to develop a deeper understanding of the attention mechanism.

Recall Equation 1, let's simplify the problem and assume $\forall\ i = 1, \cdots, N,\ x_i \in \mathbb{R}$. The Nadaraya-Watson kernel regression estimator is of the form,

$$\hat{m}(x) = \frac{\sum_{i=1}^{N} K_h(x - x_i) \cdot y_i}{\sum_{i=1}^{N} K_h(x - x_i)}$$

Using Equation 3 and Equation 4, we derive the Nadaraya-Watson kernel regression estimator like so:

$$\begin{aligned}
\mathbb{E}[Y|X = x] &= \int y f(y|x)dy = \int \frac{y f(x, y)}{f(x)}dy \\
\hat{m}(x) &= \hat{\mathbb{E}}[Y|X = x] \\
&= \int \frac{y\hat{f}(x, y)}{\hat{f}(x)}dy \\
&= \frac{\sum_{i=1}^{N} K_{h_1}(x - x_i)\int y K_{h_2}(y - y_i)dy}{\sum_{i=1}^{N} K_{h_1}(x - x_i)} \\
&= \frac{\sum_{i=1}^{N} K_{h_1}(x - x_i)\int (u + y_i) K_{h_2}(u)du}{\sum_{i=1}^{N} K_{h_1}(x - x_i)}, \text{ where } u = y - y_i \\
&= \frac{\sum_{i=1}^{N} K_{h_1}(x - x_i)(\int u K_{h_2}(u)du + y_i \int K_{h_2}(u)du)}{\sum_{i=1}^{N} K_{h_1}(x - x_i)} \\
&= \frac{\sum_{i=1}^{N} K_{h_1}(x - x_i) y_i}{\sum_{i=1}^{N} K_{h_1}(x - x_i)}
\end{aligned}$$

Note that to extend the Nadaraya-Watson kernel regression estimator such that $\mathbf{x} \in \mathbb{R}^d$, we can simply modify the formula as shown:

$$\hat{m}(\mathbf{x}) = \frac{\sum_{i=1}^{N} K_h(\|\mathbf{x} - \mathbf{x}_i\|_2) \cdot y_i}{\sum_{i=1}^{N} K_h(\|\mathbf{x} - \mathbf{x}_i\|_2)} \tag{5}$$

3 Attention

Before delving into transformers, it's essential to grasp their central mechanism: attention. We will begin by defining attention and its relationship with the Nadaraya-Watson kernel regression estimator. We will also explore the decomposition of the Scaled Dot-Product Attention, as introduced in the seminal work on Transformers by Vaswani et al. (2017). Subsequently, we will investigate various extensions of this attention mechanism.

3.1 Definition of Attention

This section aims to provide an intuitive understanding of the self-attention mechanism, followed by a formal definition.

Denote $\mathcal{D} = \{(\mathbf{k}_1, \mathbf{v}_1), (\mathbf{k}_2, \mathbf{v}_2), \cdots, (\mathbf{k}_m, \mathbf{v}_m)\}$ where $\mathcal{D}$ is a database of m tuples of key-value pairs. In a typical database scenario, a query matches a key, and the corresponding value is returned. Attention mechanisms generalize this idea by establishing a similarity measure between a query and a set of key-value pairs stored in the database, determining the weight assigned to each value. We define the attention over database D as the following:

$$\text{Attention}(\mathbf{q}, \mathcal{D}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i) \cdot \mathbf{v}_i, \tag{6}$$

where the query $\mathbf{q} \in \mathbb{R}^d$ and $\forall i = 1, \cdots, m$, the key $\mathbf{k}_i \in \mathbb{R}^d$.

Now, let's compare Equation 5 with Equation 6. Notice if we let $\mathbf{q} = \mathbf{x}$, $\mathbf{k}_i = \mathbf{x}_i$ and $\mathbf{v}_i = \mathbf{y}_i$ from Equation 5. We can rewrite the Nadaraya-Watson kernel regression estimator as follows:

$$\hat{m}(\mathbf{x}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i) \cdot \mathbf{v}_i, \text{ where } \alpha(\mathbf{q}, \mathbf{k}_i) = \frac{K_h(\|\mathbf{q} - \mathbf{k}_i\|_2)}{\sum_{i=1}^{N} K_h(\|\mathbf{q} - \mathbf{k}_i\|_2)}$$

This demonstrates that the Nadaraya-Watson kernel regression estimator is a special instantiation of attention, which marks its original, more formal appearance, predating the explicit introduction of the term *attention* (Smola & Zhang, 2019).

3.2 Scaled Dot-Product Attention

The scaled dot-product attention mechanism was proposed by Vaswani et al. (2017). We will analyze its decomposition and show its relation to kernel functions that we have studied in MA4270. The scaled dot-product attention is defined as follows:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}\left(\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}}\right) = \frac{\exp(\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}})}{\sum_{i=1}^{m} \exp(\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}})}, \tag{7}$$

where $\mathbf{q} \in \mathbb{R}^d$ and $\forall i = 1, \cdots, m, \mathbf{k}_i \in \mathbb{R}^d$.

Note that $\frac{1}{\sqrt{d}}$ is chosen as a scaling factor to compensate for the growth in dot products, which may push the softmax function to regions with extremely small gradients. To understand this choice, consider assuming that the components of $\mathbf{q}$ and $\mathbf{k}_i$ are independent random variables with $\mathbb{E}[q^{(j)}] = 0$, $\mathbb{E}[k_i^{(j)}] = 0$, $\text{Var}(q^{(j)}) = 1$, and $\text{Var}(k_i^{(j)}) = 1$. Consequently, $\mathbb{E}[\mathbf{q}^T \mathbf{k}_i] = 0$ and $\text{Var}(\mathbf{q}^T \mathbf{k}_i) = d$. To maintain the variance at 1, it is scaled by $\frac{1}{\sqrt{d}}$.

By letting $Z_1(\alpha)$ denote a normalizing constant, Song et al. (2021) show Equation 7 is further decomposed as follows:

$$\begin{aligned}\alpha(\mathbf{q}, \mathbf{k}_i) &= \frac{1}{Z_1(\alpha)} \cdot \exp(\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}}) \\ &= \frac{1}{Z_1(\alpha)} \cdot \exp(\frac{-\|\mathbf{q} - \mathbf{k}_i\|_2^2}{2\sqrt{d}}) \cdot \exp(\frac{\|\mathbf{q}\|_2^2 + \|\mathbf{k}_i\|_2^2}{2\sqrt{d}})\end{aligned} \tag{8}$$

From Equation 8, it is important to observe for a single query, $\|\mathbf{q}\|_2$ is fixed and if the keys $\mathbf{k}_i$ are normalized, e.g. passed through LayerNorm (to be discussed in subsection 4.3), the equation may be rewritten as:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{Z_1'(\alpha)} \cdot \exp(\frac{-\|\mathbf{q} - \mathbf{k}_i\|_2^2}{2\sqrt{d}}),$$

which is analogous to applying a Gaussian Kernel in the Nadaraya-Watson kernel regression estimator. More specifically, the term $\exp\left(\frac{-\|\mathbf{q}-\mathbf{k}_i\|_2^2}{2\sqrt{d}}\right)$ can be rewritten as $K_{RBF}(\mathbf{q}, \mathbf{k}_i) = \exp\left(\frac{-\|\mathbf{q}-\mathbf{k}_i\|_2^2}{2(\sqrt[4]{d})^2}\right)$, i.e., the Radial Basis Function (RBF) Kernel applied to $\mathbf{q}$ and $\mathbf{k}_i$ with a length-scale of $\sqrt[4]{d}$.

In practice, attention is usually computed for a set of queries simultaneously. Let $\mathbf{Q} = [\mathbf{q_1}, \cdots, \mathbf{q}_n]^T \in \mathbb{R}^{n \times d}$, $\mathbf{K} = [\mathbf{k_1}, \cdots, \mathbf{k_m}]^T \in \mathbb{R}^{m \times d}$ and $\mathbf{V} = [\mathbf{v_1}, \cdots, \mathbf{v}_m]^T \in \mathbb{R}^{m \times v}$. Then we can rewrite the scaled dot-product attention of multiple queries as shown:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right) \tag{9}$$

$$\mathbf{Y} = \mathbf{A}\mathbf{V} \in \mathbb{R}^{n \times v} \tag{10}$$

3.3 Kernel Trick Reloaded

The insight that the scaled dot-product attention pooling is essentially the RBF kernel suggests that we can adapt attention mechanisms to employ other kernels, analogous to the kernel trick utilized in classical machine learning algorithms. In fact, Egri & Han (2021) propose other kernels, which we will discuss here.

The linear kernel is of the form $K_{\text{Lin}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{c})^T(\mathbf{x}' - \mathbf{c})$, where $\mathbf{c}$ is a parameter specifying the origin of the non-stationary kernel. Let $\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{Z_2(\alpha)} K_{\text{Lin}}(\mathbf{q}, \mathbf{k}_i)$, where $Z_2(\alpha)$ is a normalizing constant, we obtain the following:

$$\begin{aligned}\alpha(\mathbf{q}, \mathbf{k}_i) &= \frac{1}{Z_2(\alpha)} \cdot (\mathbf{q} - \mathbf{c})^T(\mathbf{k}_i - \mathbf{c}) \\ &= \frac{1}{Z_2(\alpha)} \cdot (\mathbf{q}^T\mathbf{k}_i - \mathbf{q}^T\mathbf{c} - \mathbf{c}^T\mathbf{k}_i + \mathbf{c}^T\mathbf{c}) \\ &= \frac{1}{Z_2(\alpha)} \cdot \mathbf{q}^T\mathbf{k}_i + \frac{1}{Z_2(\alpha)} \cdot (-\mathbf{q}^T\mathbf{c} - \mathbf{c}^T\mathbf{k}_i + \mathbf{c}^T\mathbf{c}) \\ &= \frac{1}{Z_2(\alpha)} \cdot \mathbf{q}^T\mathbf{k}_i, \quad \text{assuming } \mathbf{c} = \mathbf{0} \\ &= \frac{\mathbf{q}^T\mathbf{k}_i}{\sum_{i=1}^m \mathbf{q}^T\mathbf{k}_i}, \text{ assuming } \sum_{i=1}^m \mathbf{q}^T\mathbf{k}_i \neq 0\end{aligned}$$

Another kernel is the periodic kernel with the form of $K_{\text{Per}}(\mathbf{x}, \mathbf{x}') = \exp(-\frac{2\sin^2(\pi\frac{\|\mathbf{x}-\mathbf{x}'\|_2}{p})}{l^2})$. Let $\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{Z_3(\alpha)} K_{\text{Per}}(\mathbf{q}, \mathbf{k}_i)$, where $Z_3(\alpha)$ is a normalizing constant, we obtain the following:

$$\begin{aligned}
\alpha(\mathbf{q}, \mathbf{k}_i) &= \frac{1}{Z_3(\alpha)} \cdot \exp\Big(-\frac{2\sin^2(\pi\frac{\|\mathbf{q}-\mathbf{k}_i\|_2}{p})}{l^2}\Big) \\
&= \frac{1}{Z_3(\alpha)} \cdot \exp\Big(-\frac{2\sin^2(\pi\frac{\sqrt{\|\mathbf{q}\|_2^2+\|\mathbf{k}_i\|_2^2-2\mathbf{q}^T\mathbf{k}_i}}{p})}{l^2}\Big) \\
&= \frac{1}{Z_3(\alpha)} \cdot \exp\Big(-\frac{2\sin^2(\pi\frac{\sqrt{2-2\mathbf{q}^T\mathbf{k}_i}}{p})}{l^2}\Big) \\
&= \text{softmax}\Big(-\frac{2\sin^2(\pi\frac{\sqrt{2-2\mathbf{q}^T\mathbf{k}_i}}{p})}{l^2}\Big),
\end{aligned}$$

if we normalize $\mathbf{q}$ and $\mathbf{k}_i$ such that $\|\mathbf{q}\|_2 = 1$ and $\|\mathbf{k}_i\|_2 = 1$ before calculating the attention between the query and keys.

The paper discusses additional kernels beyond this project's scope. However, it empirically demonstrates the performance of different attention variants, some of which outperform the original attention mechanism in certain tasks.

3.4 On Sequences

The kernel-related theory we have introduced so far allows us to use scaled dot-product attention as a general-purpose scoring function between two entities represented as queries and keys. In modern times, this has taken shape in the form of learning on **sequences**, an ordered collection of individual **tokens** that represent some temporal data. For example, sentences can be considered sequences with individual words as tokens. Financial time series can also be considered sequences with stock prices are tokens. Self-attention has revolutionised sequence learning by **considering tokens as queries and keys**. Vaswani et al. (2017) showcase the sequence learning aspect of self-attention through numerous language-related tasks like translation between English and German.

In the context of sequence learning, we see two variants of attention: *self-attention* and *cross-attention*. The queries, keys, values $\mathbf{q}, \mathbf{k}, \mathbf{v}$ come from the same source sequence in the former whereas the latter features keys and values $\mathbf{k}, \mathbf{v}$ from the same source sequence and queries $\mathbf{q}$ from another.

From this point forward, we focus on the sequence learning capabilities of self-attention and how the mechanism can be scaled up to improve representational power in the traditional *supervised learning* setting of Machine Learning.

3.5 Multi-Head Self-attention

To increase the complexity of self-attention, the operation in Equation 10 can be done in parallel and combined using *heads*. For a given input sequence $\mathbf{X} \in \mathbb{R}^{T\times d_{\text{in}}}$, each head $h \in \{1, \cdots, H\}$ outputs an intermediate tensor $\in \mathbb{R}^{n\times v}$ which are collectively concatenated into a tensor of shape

$\mathbf{R}^{n \times Hv}$ and projected one more time by another weight matrix W^O:

$$\mathbf{Q}^{(h)} = \mathbf{X}\mathbf{W}_{\mathbf{q}}^{(h)}, \qquad \mathbf{K}^{(h)} = \mathbf{X}\mathbf{W}_{\mathbf{k}}^{(h)}, \qquad \mathbf{V}^{(h)} = \mathbf{X}\mathbf{W}_{\mathbf{v}}^{(h)} \tag{11}$$

$$\mathbf{A}^{(\mathbf{h})} = \text{Attention}(\mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)}) \tag{12}$$

$$\mathbf{Y} = \text{concat}[\mathbf{A}^{(\mathbf{1})}, \cdots, \mathbf{A}^{(\mathbf{H})}] \cdot W^O \in \mathbb{R}^{n \times v} \tag{13}$$

Vaswani et al. (2017) find $H = 8$ to be the ideal case for sequence transduction tasks (like language translation).

4 Transformers

When individual attention layers are stacked to create columns, alongside architectural elements like feed-forward layers, residual connections, and LayerNorm, we obtain the *Transformer*. This powerful deep network has consistently outperformed benchmarks on numerous tasks across domains and continues to show promise in several others by the day. The dot-product step between queries and keys enables the transformer to ingest variable-length data in a fully parallelizable manner, compared to counterparts like RNNs and LSTMs that failed to be parallelized.

Disclaimer

We recommend finishing the prerequisite chapter on Neural Networks (Lecture Note 12) to readers from MA4270.

4.1 General Architecture

The transformer features two columns (or towers) representing the encoder and decoder, as in most sequence-to-sequence learning problems (like language translation); See Figure 1 taken directly from Vaswani et al. (2017). The encoder ingests a source sequence while the decoder takes in a target sequence. The sequences are preprocessed with *positional encodings*, a way to imbue the notion of "token order" in the sequence. Without these encodings, the transformer treats the input tokens as an unordered set, which may or may not be the underlying assumption behind the sequence structure.

Encoder. The encoder column takes in a sequence $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_{T_{\text{src}}}\}$ of fixed-size token embeddings of shape $T_{\text{src}} \times d_{\text{emb}}$. Here, T_{src} is the number of tokens in the sequence (eg: number of words in a sentence) while d_{emb} is the word embedding dimension[1]. The self-attention layers within the encoder layer ensure all input tokens adequately attend to one another, improving the representational power of the model.

Given how deep the transformer is, there is a risk of running into the *vanishing gradient problem*[2]. To alleviate this, *residual connections* are used to connect the shallow end of the encoder (before multi-head attention) to the deeper end of the encoder (after feed-forward). Incoming residual information from the shallow end is simply added to the intermediate representations at the deeper

[1] In natural language processing, words are represented by associated "word embedding" vectors that numerically represent semantic meaning. We direct interested readers to `https://lilianweng.github.io/posts/2017-10-15-word-embedding/`.

[2] The vanishing gradient problem entails gradient values shrinking towards zero during backpropagation due to repeated multiplications of small numbers. It is especially common in very deep models with several layers.

end before being sent out. In doing so, these connections offer additional "highways" to pass information from the shallow end of the encoder to the deeper end, ensuring the coherence of the input sequence throughout the encoder stack. During backpropagation, the gradients can take these "highways", allowing for smoother training and no more vanishing gradients. On its own, the encoder layer can be used for classification and regression tasks where only input sequences are part of the dataset.

Decoder. The decoder functions similarly but ingests a target sequence of shape $T_{\text{tgt}} \times d_{\text{emb}}$ that represents the desired output of the entire transformer (e.g., a sentence in another language). Similarly, positional encodings are applied to the target sequence and fed into the stack of self-attention layers. The decoder features cross-attention layers that ingest the final keys $\mathbf{K}$ and values $\mathbf{V}$ from the encoder column. This allows the decoder to attend to parts of the input sequence and how it influences the structure of the target sequence. Additionally, like the encoder, the decoder also has residual connections to maintain the quality of the intermediate representations inside the network. The decoder is used for sequence generation by ingesting a sequence of tokens and predicting the next tokens *autoregressively*, i.e., one token at a time using the previously generated tokens as context.

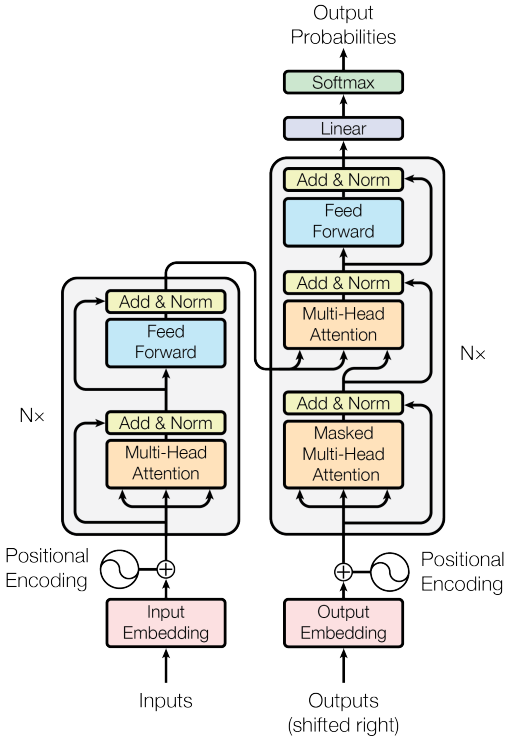

Figure 1: The Transformer layer comprising an encoder and decoder. Source sequences are fed into the encoder while target sequences are fed into the decoder.

Decoder masking. Pertinently, during inference or sampling, the decoder uses *triangular masking* to prevent the attention mechanism from seeing future tokens (i.e., any word to the right of another). Specifically, when the pre-softmax attention matrix $\mathbf{QK}^T/\sqrt{d} \in \mathbb{R}^{T\times T}$ is computed, a mask with a lower triangle of 1 and upper triangle of $-\infty$ is applied. At each token index $t \in \{1, \cdots, T\}$ along the sequence, masking ensures only tokens from $1 \to t-1$ are visible to the token at t. When the $-\infty$ values in the upper triangular part of $\mathbf{QK}^T/\sqrt{d}$ are put through the softmax, they become 0, essentially preventing the model from "cheating". For instance, during sentence generation tasks, only words to the left of the current word are considered when predicting the next word, preventing the model from "peeking" at the correct prediction (which it has access to).

In the vanilla transformer introduced by Vaswani et al. (2017), there are two encoder and two decoder layers, with the final keys and values from the topmost encoder feeding into each decoder layer separately. This ensures embedding information from the encoder is sustained at the shallow and deep ends of the decoder, maintaining the coherence of the source and target sequences.

4.2 Positional Encodings

Existing work has shown the efficacy of positional encodings in aiding transformers to learn rich representations over input sequences. These encodings are often added or concatenated to token embeddings. Positional encodings provide a notion of *position* or *order* in sequence space, allowing the transformer to learn relative token distance and sequence structure. In Vaswani et al. (2017), sinusoidal positional encodings offer an efficient way to show relative order between tokens. Suppose t is the token index in a sequence, the final encoding $\vec{p_t}$ is created by passing the token index into $f(t)$ which defines the elements (i) in the vector based on odd and even positions as follows:

$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k t), & \text{if } i = 2k \quad \text{(even vector index)} \\ \cos(\omega_k t), & \text{if } i = 2k+1 \quad \text{(odd vector index)} \end{cases} \tag{14}$$

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} t) \\ \cos(\omega_{d/2} t) \end{bmatrix} \in \mathbb{R}^d \tag{15}$$

If d is the embedding dimension, i refers to sub-indices through this embedding dimension (i.e., the real-valued elements of the positional encoding vector). Thereafter, $\vec{p_t}$ is either added or concatenated to the corresponding token embedding $\mathbf{x}_t$. This positional encoding computation step is done only once before being passed into the encoder or decoder.

Vaswani et al. (2017) also mentions this function is hypothesized to enable the model to attend by relative positions, as for any fixed offset δ, the positional encoding at position $t + \delta$ can be represented by a linear projection of that at position t. Observe for any fixed offset δ:

$$\begin{aligned} \begin{bmatrix} \cos(\delta\omega_i) & \sin(\delta\omega_i) \\ -\sin(\delta\omega_i) & \cos(\delta\omega_i) \end{bmatrix} \begin{bmatrix} \vec{p_t}^{(2i)} \\ \vec{p_t}^{(2i+1)} \end{bmatrix} &= \begin{bmatrix} \cos(\delta\omega_i)\vec{p_t}^{(2i)} + \sin(\delta\omega_i)\vec{p_t}^{(2i+1)} \\ -\sin(\delta\omega_i)\vec{p_t}^{(2i)} + \cos(\delta\omega_i)\vec{p_t}^{(2i+1)} \end{bmatrix} \\ &= \begin{bmatrix} \cos(\delta\omega_i)\sin(\omega_i t) + \sin(\delta\omega_i)\cos(\omega_i t) \\ -\sin(\delta\omega_i)\sin(\omega_i t) + \cos(\delta\omega_i)\cos(\omega_i t) \end{bmatrix} \\ &= \begin{bmatrix} \sin(\omega_i(t+\delta)) \\ \cos(\omega_i(t+\delta)) \end{bmatrix} \\ &= \begin{bmatrix} \vec{p_{t+\delta}}^{(2i)} \\ \vec{p_{t+\delta}}^{(2i+1)} \end{bmatrix} \end{aligned}$$

4.3 Layer Normalization

In the transformer architecture, the Add & Norm layer (see Figure 1) employs residual connections followed by layer normalization, or LayerNorm, as shown below:

$$\text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x})) \tag{16}$$

Here, Sublayer($\mathbf{x}$) represents the function implemented by the layer that precedes it. It's worth noting that there is literature suggesting an alternative placement of LayerNorm, advocating for it to precede the Sublayer (Xiong et al., 2020).

At its core, the idea of LayerNorm is to perform normalization across the dimension of each data point. Given an input $\mathbf{x} \in \mathbb{R}^d$, it computes:

$$\text{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \boldsymbol{\mu}}{\sigma} \tag{17}$$

where $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$, $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$, and $\boldsymbol{\mu} = [\mu, ..., \mu] \in \mathbb{R}^d$.

However, the normalization of keys through LayerNorm is crucial to the expressivity of the subsequent multi-head attention layer (Brody et al., 2023). We will illustrate the geometric effect of LayerNorm on the keys and provide an intuitive understanding of why it enhances expressivity.

To begin, we will demonstrate that for $\mathbf{x} \in \mathbb{R}^d$, $\hat{\mathbf{x}}$ is orthogonal to $\vec{\mathbf{1}}$ and $||\hat{\mathbf{x}}||_2 = \sqrt{d}$, where $\hat{\mathbf{x}} = \text{LayerNorm}(\mathbf{x})$ and $\vec{\mathbf{1}} = [1, ..., 1]^T \in \mathbb{R}^d$.

$$\begin{aligned}(\mathbf{x} - \boldsymbol{\mu}) \cdot \vec{\mathbf{1}} &= \mathbf{x} \cdot \vec{\mathbf{1}} - \boldsymbol{\mu} \cdot \vec{\mathbf{1}} \\ &= \sum_{i=1}^{d} x_i - (\frac{1}{d}\sum_{i=1}^{d} x_i) \cdot d \\ &= 0\end{aligned} \tag{18}$$

$$\begin{aligned}||\hat{\mathbf{x}}||_2 &= \frac{1}{\sigma} \cdot ||\mathbf{x} - \boldsymbol{\mu}||_2 \\ &= \frac{\sqrt{d}}{\sigma} \cdot \sqrt{\frac{1}{d} \cdot \sum_{i=1}^{d}(x_i - \mu)^2} \\ &= \frac{\sqrt{d}}{\sigma} \cdot \sigma = \sqrt{d}\end{aligned} \tag{19}$$

With these two properties, we observe that LayerNorm can be decomposed as a projection onto a hyperplane $\mathcal{H}$ that is perpendicular to $\vec{\mathbf{1}}$, as well as a scaling of the projected vectors to a norm of $\sqrt{d}$.

Intuitively, this geometric structure can enhance the learning capability of the transformer, as demonstrated by Brody et al. (2023). They show that such a projection can help the attention mechanism equally attend to keys, while scaling the projected vectors can minimise the risk of *unselectable keys* (when attention fails to assign the highest score to the key). While the proof is beyond the scope of our current project, we will provide intuition on why this occurs.

Figure 2 depicts a visualization of the attention mechanism with and without LayerNorm[3]. Recall from Equation 11 that for $\mathbf{A^{(h)}}$, the queries $\mathbf{Q}$ are first projected by the learnable parameter matrix $\mathbf{W_q^{(h)}}$ and the keys are projected by the learnable parameter matrix $\mathbf{W_k^{(h)}}$. Note that the attention scoring function can now be written as:

$$\frac{(\mathbf{Q}\mathbf{W_q^{(h)}})(\mathbf{K}\mathbf{W_k^{(h)}})^T}{\sqrt{d}} = (\frac{\mathbf{Q}\mathbf{W_q^{(h)}}(\mathbf{W_k^{(h)}})^T}{\sqrt{d}})\mathbf{K}^T$$

From this point, we will refer to $(\frac{\mathbf{Q}\mathbf{W_q^{(h)}}(\mathbf{W_k^{(h)}})^T}{\sqrt{d}})$ as the projected queries and $\mathbf{K}$ as the keys. Two important observations emerge from this visualization:

[3]Image taken from Brody et al. (2023).

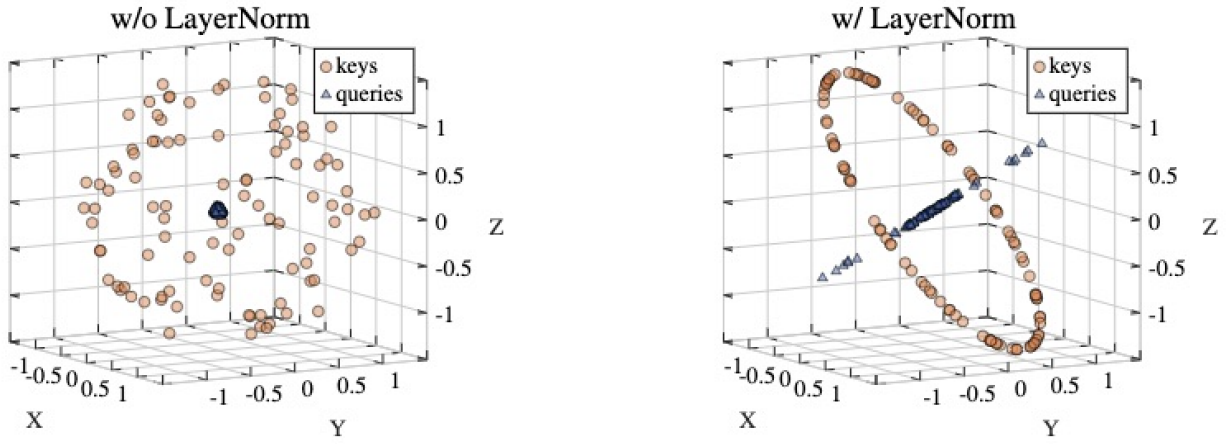

Figure 2: The effect of LayerNorm on queries and keys. **Takeaway:** Using LayerNorm geometrically modifies the structure of the keys which helps improve learning.

1. To attend to keys with equal scores, the projected queries tend to align more with $\vec{\mathbf{1}}$, i.e., they become almost orthogonal to the keys.

2. All projected keys are "selectable", meaning they can receive the maximal attention score, as they lie within the convex hull formed by the keys.

These observations provide insight into how LayerNorm improves the expressivity of transformers. The paper also empirically demonstrates the faster convergence and better generalization with LayerNorm.

5 Experiments

Here, we explore the efficacy of transformers as sequence learners on a collection of periodic functions. We observe whether the model can capture patterns across time and believe periodic functions to be an ideal demonstration of learning on *dynamic* information.

Choice of functions. We select the following 3 periodic functions with varying behavior to model:

$$1.\ \ y = \sin(x) \tag{20}$$
$$2.\ \ y = \sin(x) \cdot \exp(0.01x) \tag{21}$$
$$3.\ \ y = \text{square}(x) \tag{22}$$

All functions have the same period of 2π. The second function is essentially the first function with an exponential modification to amplify it. The third function can be considered a discrete version of the first, and may be non-differentiable in certain regions, offering the additional challenge of not implicitly learning gradient (as in, *slope*) information to predict timesteps.

Evaluation. We measure model performance between predicted and actual values on the curves using a mean squared error (MSE).

Setup. We use a transformer encoder with 2 layers, each with $H = 2$ heads, and train for 100 epochs. We project our input sequences of shape $(T_{\text{src}}, 1)$ to a hidden dimension of $d_{\text{emb}} = 512$. We use positional encodings of dimension 128.

Modifications to Transformer. In Section 4, we describe the decoder and how it helps with sequence generation using triangle masking. However, we choose to bypass the decoder with a single **fully-connected layer** because we are predicting the next token instead of generating sequences. Formally,

$$\text{FullyConnected}(\mathbf{x}) = \mathbf{x}\mathbf{W}^T + \mathbf{b}, \tag{23}$$

where $\mathbf{W}, \mathbf{b}$ are trainable parameters. This is akin to performing simple linear regression (Chapter 4 in MA4270) on the input data $\mathbf{x} \in \mathbb{R}^d$. This layer has other synonyms such as Dense layer and Feed-forward layer.

5.1 Results

Learning periodic functions. We first train the transformer on short sequence fragments from the dataset for each function (of sequence length $T_{\text{src}} = 100$). We then predict 200 future timesteps during the inference stage. Table 1 reports the average MSE values across the predicted timestep values. All the functions are learned well enough for the MSE values to be close to 0, indicating the model has implicitly modelled the dynamics of each system.

Function	**Test MSE**
$y = \sin(x)$	0.0050
$y = \sin(x) \cdot \exp(0.01x)$	0.0085
$y = \text{square}(x)$	0.0243

Table 1: Evaluation on unseen inputs for the three functions. **Takeaway:** Transformers are capable of modelling periodic functions precisely.

Figure 3 visualises the predicted timestep values using our trained transformer for each function. The transformer can learn dynamics information even when the wave is not behaving the same across periods of 2π, as seen by the middle figure.

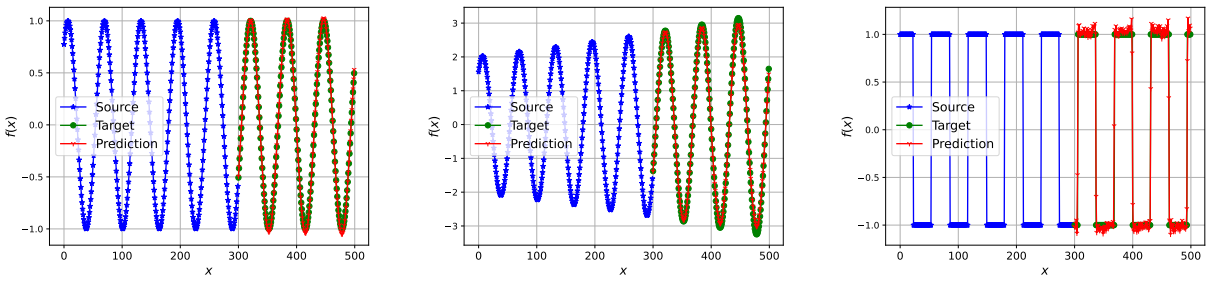

Figure 3: Visualisation of predicted timesteps (Prediction, red) overlaid with training samples (Source, blue), future targets (Target, green). Left: Sinusoidal wave. Middle: Increasing sinusoidal wave. Right: Square wave. **Takeaway:** Our model can learn dynamic systems with minimal error.

Analysing attention weights. Here, we visualise the (scaled dot-product) self-attention map $\mathbf{A}^{(h)}$ across the $H = 2$ heads to observe what parts (tokens) of the sequence the model focuses on when predicting the next tokens. Figure 4 shows these attention matrices for the first head in the second layer.

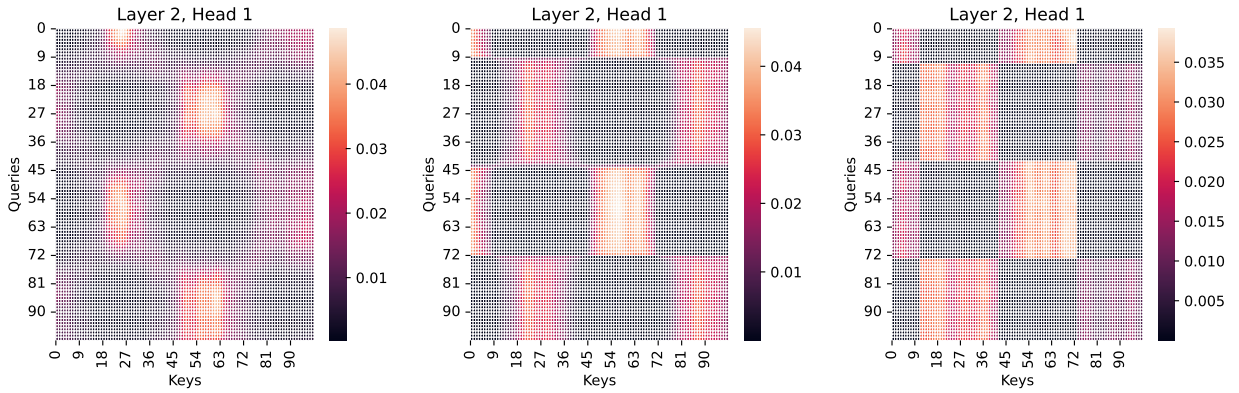

Figure 4: Visualisation of attention maps $\mathbf{A}^{(1)}$ from the second layer's first head. Left: Sinusoidal wave. Middle: Increasing sinusoidal wave. Right: Square wave. **Takeaway:** The periodic nature of the data is reflected in the hotspots on the heatmap, showing our model picks up on the periodic signal.

The checkerboard nature of the attention maps suggests the model is implicitly learning the periodic nature of the functions. For a given query, the hotspots occur at keys with a $\sim$60 timestep difference which corresponds to the predetermined period of 2π during data generation. Likewise, for a given key, the hotspots occur horizontally in a similar manner. This means the heatmap is also roughly symmetric since we are using self-attention; the queries and keys are roughly similar, causing the checkerboard pattern to repeat along both queries and keys.

Moreover, aside from the periodicity, the transformer's heatmap also reflects the rate of change of $y = f(\cdot)$ from peak to trough. For instance, the hotspots for the square wave are more well-defined than the sinusoidal wave, because the rate of change of the square wave from peak to trough is more extreme.

Altogether, our experiments show that transformers are capable sequence learners adept at capturing patterns in the underlying data distribution. The analysis suggests that transformers implicitly understand the notion of "order" in a sequence, making them powerful models to use in real-world settings.

6 Conclusion

In this report, we explain the inner workings of the Transformer which has shown great potential for various ML tasks. We have shown how the attention mechanism was historically motivated by the kernel machines literature, specifically the Nadaraya-Watson kernel regression estimator. The transformer can then be considered a generalization of the Nadaraya-Watson kernel regression estimator, which uses a scoring function to attend to tokens appropriately for sequence learning problems. In rigorously explaining this connection, we have also shown the efficacy of the transformer on toy data comprising periodic functions, showcasing its effectiveness and justifying its widespread utility.

References

Shaked Brody, Uri Alon, and Eran Yahav. On the expressivity role of layernorm in transformers' attention. *arXiv preprint arXiv:2305.02582*, 2023.

Gokhan Egri and Xinran Nicole Han. Attention is kernel trick reloaded. 2021.

Alex Smola and Aston Zhang. Attention in deep learning. 2019.

Kyungwoo Song, Yohan Jung, Dongjun Kim, and Il-Chul Moon. Implicit kernel attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9713–9721, 2021.

Ryan Tibshirani. Kernel regression, January 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020.

A Appendix

Additional Material

Aside from our main focus on Transformers by Vaswani et al. (2017), we additionally explore the role of LayerNorm inside the transformer architecture by further introducing theory from Brody et al. (2023). **We were given explicit approval** by Prof. Soh to take up this project despite it being out of scope.

Author Contributions

RA conceived the study on transformers and LayerNorm. RCYS introduced the theory for kernels and connections to self-attention. RA wrote the sections on transformers and positional encodings. RCYS wrote about LayerNorm. RA and RCYS jointly implemented the data loaders, model, and helper functions. RA implemented the plotting code and attention visualisations. RA and RCYS jointly analysed the outputs from the model.

Source Code

We open-source our project here: `https://github.com/rish-16/ma4270-project`. We were also given approval to use PyTorch to implement our models from scratch.

We also credit the following resources:

- Tutorial on Transformers for architectural details: `https://pytorch.org/tutorials/beginner/transformer_tutorial.html`
- Tutorial on Transformers for helper methods to visualise the attention maps: `https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html`.
- Guide on interpreting attention maps for transformers: `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15794705.pdf`.