UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Efficient transformers applied to video classification

Autor: Oriol Martínez Pérez

Director:	**Sergio Escalera**
Codirectors:	**Albert Clapés**
	David Pujol
Realitzat a:	**Departament de Matemàtiques i Informàtica**
Barcelona,	**12 de juny de 2023**

Contents

1 Introduction **1**
1.1 Objectives . . . 3

2 Self-attention: a mathematical construction **5**
2.1 Kernels . . . 5
2.2 Self-attention . . . 8
2.2.1 Computational complexity of self-attention . . . 11

3 Efficient transformers **12**
3.1 Attention reformulations . . . 13
3.1.1 Cosformer . . . 13
3.2 Low-rank . . . 15
3.2.1 $\mathcal{S}$ is low rank . . . 15
3.2.2 Linformer . . . 18
3.2.3 Nyströmformer . . . 19

4 Transformer architecture **23**
4.1 Input embedding . . . 23
4.1.1 Positional encoding . . . 25
4.2 Attention Block . . . 26
4.2.1 Multi-Head Attention . . . 26
4.2.2 Feed forward . . . 27
4.2.3 Linear Classifier . . . 28

5 Experimental comparison between self-attention mechanisms **29**
5.1 Dataset . . . 30
5.1.1 Clips preprocessing . . . 31
5.2 Experimental Setup . . . 32
5.3 Results . . . 34

6 Conclusions **41**

Abstract

Transformers, with the self-attention mechanism on its core, have shown great performance on several Machine Learning areas such as NLP or Computer Vision since its appearance at 2017 [1]. However, its quadratic time and memory complexity on the input length makes its application prohibitive when dealing with large input sequences. This motivated the appearance of several self-attention reformulations in order to lower its complexity and make its development less costly.

We focus on three of these self-attention mechanisms applied to video classification: Cosformer [2], Nyströmformer [3] and Linformer [4]. Concretely, our goal in this project is to suggest which of them is best suited for this task. To evaluate each model performance, we design a personalizable Transformer with interchangeable self attention mechanisms and train it using a simplified dataset derived from EpicKitchens-100 [5]. We carefully describe the Transformer architecture, explaining the purpose of each of its modules, and provide and overall description of how internally works. Preliminary results indicate that Nyströmformer is the best option, being the model which converged faster and achieved the best trade off between computational cost and classification metrics. Linformer obtained similar results and Cosformer apparently failed to perform the classification.

The theoretical formalization of the aforementioned self-attention mechanisms is essential for their results interpretation. Hence, we also provide an in-depth mathematical description of both the original self-attention mechanism presented by Vaswani [1] and the three efficient mechanisms. We realize a complexity analysis of all mechanisms and expose its main properties, linking the theoretical basis with the results.

All code is publicly available on GitHub.

2020 Mathematics Subject Classification. 68T45, 68T50

Acknowledgments

I wanted to express my gratitude to Sergio Escalera for his guidance and positive feedback during all this work. Also, I am really grateful to Albert Clapés and David Pujol for their constant corrections and support. Special thanks to David for helping me with the code.

Finally, I sincerely want to thank my parents for always being by my side.

Chapter 1

Introduction

Throughout the last years, Machine Learning has had an unprecedented growth, increasing its efficiency and performance in several areas. Convolutional Neural Networks (CNNs) [6] and Recurrent Neural Networks (RNNs) [7] were the way to go in a large variety of tasks during the last decade until the proposal of a new architecture in 2017 disrupted the paradigm: the **Transformer** [1]. Nowadays, it is the core of famous Machine Learning models such as ChatGPT and has replaced CNNs and RNNs on different areas such as NLP and computer vision. But before going to further detail, we find it critical to comprehend what transformers are and why its appearance had such influence.

The first Transformer [1] was designed to deal with NLP tasks. These tasks handle language related problems, such as translation or summarizing [8], which have a strong sequential behaviour. It has been found [9] that in order to effectively solve NLP tasks, Machine Learning models must carefully consider the context of words and sentences.

Traditional CNNs lack the ability to do that, motivating the appearance of RNNs, which added inner connections among the different layers of a network to provide previous information. In fact, recent works have found that the combination of both CNNs and RNNs perform well, with CNNs focusing on feature extraction and RNNs handling the sequentially of language [10]. However, two main drawbacks hinder the performance of this approach.

On the one hand, CNNs have inductive biases of locality [11], meaning that they exclusively rely on short range connections. For instance, a word at the begging of a text may be connected to a word in the last sentence, but a convolutional layer would only focus on their surrounding words. Therefore, this bias restricts long range connections and may impact the model performance. On the other hand, RNNs behave sequentially providing information from previous inputs, but are not able to access further inputs. Moreover, when inputs are large, RNNs tend

to ignore information distant from the current one. Hence, the model cannot fully contextualize the whole text, degrading its performance.

The Transformer presented by Vaswani [1] surpassed the two aforementioned drawbacks. By relying on self attention, it allowed to consider all the pair-wise relations between words at once and removed recurrence.

But what is self attention? To illustrate the main idea behind it, we provide an example. Suppose that we want to perform some NLP task with the text *Transformers are awesome. I love them!* An estimation of how each word is related to the others is our objective. A straightforward way to consider these relationships is with a matrix:

	Transformers	*are*	*awesome*	*I*	*love*	*them*
Transformers	1	0.7	0.6	0.2	0.3	0.8
are	0.7	1	0.8	0.2	0.1	0.6
awesome	0.6	0.8	1	0.2	0.3	0.3
I	0.2	0.2	0.2	1	0.8	0.6
love	0.3	0.1	0.3	0.2	1	0.8
them	0.8	0.6	0.3	0.6	0.8	1

(1.1)

The matrix coefficients lay in a fixed range, [0,1] in this example, and estimate how strong is the influence of a word with another. Notice how, for example, *Transformers* and *them* have a large coefficient since they are highly correlated, while *love* and *are* do not.

Transformers use self attention as their core mechanism to build this matrix and perform a desired task. Regardless of being first contemplated for NLP tasks, Transformers have been applied to other Machine Learning fields based on the same idea: constructing a matrix expressing the relationship among the inputs. Its effectiveness has been tested through last years and it has become the gold standard for several tasks. Nevertheless, self attention has a large drawback. It is computationally very expensive, yielding a quadratic complexity on the input length, which becomes prohibitive in scenarios where the input size is in the order of thousands.

In this work, an extensive overview of self attention is provided, focusing on dealing with its complexity problem. Concretely, we provide both a theoretical and experimental analysis to present and test several representative techniques that attempt to overcome it.

1.1 Objectives

The main goals of this project are two-fold:

The first consists on mathematically constructing and defining the self-attention mechanism. A theoretical basis of the necessary concepts is provided, eventually building up the self attention mechanism defined by [1]. Also, an analysis over its computational cost is reported to expose its main drawback. Once we have a deep understanding of self attention, we explain some of the main methods to overcome its complexity issue. We carefully describe the ideas and concepts behind them, and present three of the most common state of the art mechanisms: the Cosformer [2], the Linformer [4] and the Nyströmformer [3]. A deep explanation of the techniques used by this methods is presented together with an analysis of their computational cost. All this is in Chpt. 2 and Chpt. 3, being the most theoretical part of this work.

The second objective is to build up a Transformer from scratch to solve a real task, and compare how the efficient methods presented at Chpt. 3 behave.

Concretely, we use self attention to classify videos according to [12], which can be easily treated as large inputs and are well suited to analyze the complexity problem, as described at Chpt. 4. Moreover, videos imply the additional difficulty of handling long term dependencies, being an inherent consequence of videos having a temporal dimension. Therefore, our objective is to answer the following question: *Which self attention mechanism has the best performance/computational cost trade off for video classification?*

To fulfill this objective, in Chpt. 5 we construct and train our model using PyTorch [13] and a dataset derived from EpicKicthens-100 [5]. The dataset consists on first-person videos of kitchen activities and a simplified version is generated from the original one in order to reduced our experiment complexity. Our classification task is to classify the verb of the action taking place in the video. We train the model with the aforementioned dataset and the three chosen self attention mechanisms, together with the original self attention as our baseline, and compare them. The criteria we use to determine their efficiency is the computation of the Matthews coefficient together with the number of FLOPs required to compute self attention and the memory required during training. A graphic, available at Fig. 5.2, shows the obtained results.

The results presented at Sect. 5.3 indicated that Nyströmformer is the most efficient mechanism. It is the only model that fully converged in our experiment and achieved the highest Matthews coefficient. Moreover, its memory usage was the lowest among all the tested mechanisms. Linformer achieved similar results and its convergence was not fully achieved, suggesting that it may surpass Nyströmformer with larger training. On the other hand, Cosformer results were clearly

inferior, probably due to a bad optimization. Nevertheless, we discuss some theoretical explanations which could justify its bad metrics.

The implementation of this project is publicly available on GitHub.

Chapter 2

Self-attention: a mathematical construction

In this chapter a fully mathematical construction of self attention is presented. The goal is to achieve a function that given an input set, produces a matrix similar to (1.1), which expresses the relationships among the inputs.

The concept of *kernel* is the basis selected to construct it [14], and it is carefully described. The theoretical basis of kernels, stating some of their properties, are presented and used to generate the self attention function designed at [1]. To conclude, self attention complexity is analyzed, exposing which calculations are cumbersome and the implications that arise from them.

Notation: During all this chapter and the following ones, we recurrently work with matrices. From now on, we consider all matrices to be real, and we denote them as $M_{n\times m}$, with n and m indicating the number of rows and columns respectively.

2.1 Kernels

We begin by describing what kernels are and the properties that will be used during all this work to treat self-attention. All kernels theory is extracted from [15].

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be any set, $n \leq \infty$.

Definition 2.1. Given $x, y \in \mathcal{X}$, a **kernel** is a function

$$k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$$

where $k(x, y)$ is a real number characterizing their pairwise similarity.

Definition 2.2. Let k be a kernel of $\mathcal{X}$.
Its **Gram matrix**, G, is defined as

$$G = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{pmatrix} \tag{2.1}$$

We say that a kernel k is **positive semidefinite** if G is a **positive semidefinite** matrix. I.e. if G satisfies one of theses conditions:

a) $\exists M \in M_{n\times n}$ such that $G = M^\top M$

b) $x^\top G x \geq 0$, $\forall x \in \mathbb{R}^n$

c) The eigenvalues of G are non negative.

Until now, there is not any restriction on our set. In order to ensure that we can define correctly a kernel, it is convenient to map our set to a suitable space where computations can be calculated without problems.

Definition 2.3. Let $\mathcal{H}$ be a linear space equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. A **feature map** is a function ϕ

$$\phi : \mathcal{X} \longrightarrow \mathcal{H}$$

From now on, we choose $\mathcal{H}$ to be the Euclidean space, $\mathcal{H} = \mathbb{R}^d$, with the usual inner product:

$$\langle x, y \rangle_{\mathbb{R}^d} = \sum_{i=1}^{d} x_i \cdot y_i$$

where $x, y \in \mathbb{R}^d$. Geometrically speaking, if x and y are normalized to 1, it gives us the cosine of the angle between both vectors, which can be understood as a similarity measure. This allows us to define a kernel after selecting a suitable feature map for our set $\mathcal{X}$.

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathbb{R}^d} \tag{2.2}$$

for $1 \leq i, j \leq n$. We denote $\langle \cdot, \cdot \rangle_{\mathbb{R}^d}$ as $\langle \cdot, \cdot \rangle$ to soften notation.

Proposition 2.4. *The kernel $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ defined by (2.2) is positive semidefinite.*

Proof. Let $z \in \mathbb{R}^n$. Let's check that it satisfies condition b) of Def. 2.2.

$$z^\top G z = \sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j \langle \phi(x_i), \phi(x_j) \rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} \langle z_i \phi(x_i), z_j \phi(x_j) \rangle = \| \sum_{i=1}^{n} z_i \phi(x_i) \|^2 \geq 0$$

where $\| \cdot \|$ is the usual norm defined by $\| \cdot \| = \langle \cdot, \cdot \rangle^{1/2}$. □

At some point, we may want to combine different kernels to generate new ones. However, we are interested in conserving the positive semidefiniteness of their Gram matrices, since it warranties us some conditions that will be necessary for further calculations. The next propositions ensures us which combinations can be done without losing this property.

Proposition 2.5. *Let A and B be two positive semidefinite matrices. Then, their element-wise multiplication is positive semidefinite.*

Proof. Suppose that $A, B \in M_{n\times n}$. Let $D \in M_{n\times n}$ be a diagonal matrix generated from the eigenvalues λ_i, $1 \leq i \leq n$, of A. Since A is positive semidefinite, by definition $\lambda_i \geq 0$. Consider an orthogonal matrix $U \in M_{n\times n}$ whose column vectors, U_i, are unit eigenvectors that are orthogonal to each other. We can express A as

$$A = UDU^\top = \sum_{i=1}^{n} \lambda_i U_i U_i^\top \tag{2.3}$$

Analogously, if $\mu_i, v_i, 1 \leq i \leq n$, are the eigenvalues and eigenvectors of B, respectively, then we can express B as

$$\sum_{i=1}^{n} \mu_i V_i V_i^\top \tag{2.4}$$

We have

$$\begin{aligned}\left(U_i U_i^\top\right) \cdot \left(V_j V_j^\top\right) = \left(u_{i,k}u_{i,l} \cdot v_{j,k}v_{j,l}\right)_{k,l} &= \left(u_{i,k}v_{j,k} \cdot u_{i,l}v_{j,l}\right)_{k,l} \\ &= \left(U_i \cdot V_j\right)\left(U_i \cdot V_j\right)^\top \quad 1 \leq h,k \leq n\end{aligned} \tag{2.5}$$

Writing $U_i \cdot V_j = [y_1, \ldots, y_n] \in \mathbb{R}^n$, then component (h,l) of $\left(U_i \cdot V_j\right)\left(U_i \cdot V_j\right)^\top$ is $y_h y_l$. Therefore,

$$\sum_{h=1}^{n}\sum_{l=1}^{n} z_h z_l y_h y_l = \left(\sum_{h=1}^{n} z_h y_h\right)^2 \geq 0, \quad \forall z_1 \ldots z_n \tag{2.6}$$

We proofed that the matrix obtained at (2.5) is positive semidefinite. Since $\lambda_i, \mu_j \geq 0\ \forall i, j$ by definition,

$$A \cdot B = \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \mu_j \left(U_i U_i^\top\right) \cdot \left(V_j V_j^\top\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \mu_j \left(U_i \cdot V_j\right)\left(U_i \cdot V_j\right)^\top \tag{2.7}$$

is positive semidefinite. □

Proposition 2.6. *Let $k_1, k_2, \dots$ be positive semidefinite kernels of $\mathcal{X}$. The following kernels are also positive semidefinite*:

1. $ak_1 + bk_2$, *with* $a, b \in \mathbb{R}^+$
2. $k_1 k_2$ *(multiplication)*

Proof. For $A, B \in M_{n\times n}$ the Gram matrices of k_1 and k_2 respectively, the first property follows from

$$x^\top Ax \geq 0, \quad x^\top Bx \geq 0 \Rightarrow x^\top (aA + bB)\, x \geq 0, \quad \forall x \in \mathbb{R}^n \tag{2.8}$$

The second is directly proof using Prop. 2.5. □

2.2 Self-attention

We can now define the self-attention mechanism [1]. As before, let $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ be any set, $n \leq \infty$. Our objective is to create a kernel which characterize the similarity between the elements of $\mathcal{X}$, and use it to perform a specified task.

First, we define 3 different feature maps which send an element $x_i \in \mathcal{X}$ to $\mathbb{R}^d$:

$$\begin{array}{ccc} W_Q : \mathcal{X} \longrightarrow \mathbb{R}^d & W_K : \mathcal{X} \longrightarrow \mathbb{R}^d & W_V : \mathcal{X} \longrightarrow \mathbb{R}^d \\ x_i \mapsto x_q^i & x_i \mapsto x_k^i & x_i \mapsto x_v^i \end{array}$$

Notice that if $\mathcal{X} = \mathbb{R}^n$, then W_Q, W_K and W_V are matrices which changes the vectors dimensionality.

Now, we can apply W_Q to all elements of $\mathcal{X}$ and put them together as a matrix:

$$Q = \begin{pmatrix} (x_q^1)_1 & (x_q^1)_2 & \dots & (x_q^1)_d \\ (x_q^2)_1 & (x_q^2)_2 & \dots & (x_q^2)_d \\ \vdots & \vdots & & \vdots \\ (x_q^n)_1 & (x_q^n)_2 & \dots & (x_q^n)_d \end{pmatrix}$$

Analogously, we can do the same with W_K and W_V.

$$K = \begin{pmatrix} (x_k^1)_1 & (x_k^1)_2 & \dots & (x_k^1)_d \\ (x_k^2)_1 & (x_k^2)_2 & \dots & (x_k^2)_d \\ \vdots & \vdots & & \vdots \\ (x_k^n)_1 & (x_k^n)_2 & \dots & (x_k^n)_d \end{pmatrix} \quad V = \begin{pmatrix} (x_v^1)_1 & (x_v^1)_2 & \dots & (x_v^1)_d \\ (x_v^2)_1 & (x_v^2)_2 & \dots & (x_v^2)_d \\ \vdots & \vdots & & \vdots \\ (x_v^n)_1 & (x_v^n)_2 & \dots & (x_v^n)_d \end{pmatrix}$$

Definition 2.7. We denote **query**, **key** and **value** to the $\mathfrak{m}_{n\times d}$ matrices **Q**, **K** and **V** defined above. [1]

The construction of the self-attention begins with the element wise multiplication of Q and $K^\top$:

$$QK^\top = \begin{pmatrix} \langle x_q^1, x_k^1\rangle & \langle x_q^1, x_k^2\rangle & \dots & \langle x_q^1, x_k^n\rangle \\ \langle x_q^2, x_k^1\rangle & \langle x_q^2, x_k^2\rangle & \dots & \langle x_q^2, x_k^n\rangle \\ \vdots & \vdots & & \vdots \\ \langle x_q^n, x_k^1\rangle & \langle x_q^n, x_k^2\rangle & \dots & \langle x_q^n, x_k^n\rangle \end{pmatrix} = \begin{pmatrix} k(x_q^1, x_k^1) & k(x_q^1, x_k^2) & \dots & k(x_q^1, x_k^n) \\ k(x_q^2, x_k^1) & k(x_q^2, x_k^2) & \dots & k(x_q^2, x_k^n) \\ \vdots & \vdots & & \vdots \\ k(x_q^n, x_k^1) & k(x_q^n, x_k^2) & \dots & k(x_q^n, x_k^n) \end{pmatrix}$$

where we used (2.2). Therefore, $QK^\top$ gives us a first estimation of how similar the queries and their corresponding keys are with each other, since its coefficients are the output of a kernel function.

Definition 2.8. We refer to the coefficients of the matrix $S = QK^\top$ as **scores**, and we denote each of its elements by s_{ij}.

Nevertheless, s_{ij} are real values that can range in a large interval and is desirable to normalize them, since it boosts performance during training, avoiding excessively small gradients [1]. Let's focus on a single score s_{ij}. By definition, we have that

$$s_{ij} = \sum_{j=0}^{d} (x_q^i)_j \cdot (x_k^i)_j \tag{2.9}$$

We can consider that each entry of Q and K have a random distribution with mean 0 and variance 1. However, recall that both entries are independent of each other. Now, using that for two independent and randomly distributed variables X and Y:

- $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$
- $Var(X+Y) = Var(X) + Var(Y)$
- $Var(XY) = (Var(X) + \mathbb{E}[X]^2) + (Var(Y) + \mathbb{E}[Y]^2) + \mathbb{E}[X]^2\mathbb{E}[Y]^2$

We have that

$$Var((x_q^i)_j \cdot (x_k^i)_j) = Var((x_q^i)_j)Var((x_k^i)_j) = 1 \tag{2.10}$$

[1] It is common for V to life in $M_{n\times d_v}$, with $d_v \neq d$. In this work, however, for sake of simplicity we fix $d_v = d$.

Therefore,

$$Var(s_{ij}) = \sum_{j=0}^{d} Var((x_q^i)_j \cdot (x_k^i)_j) = d \tag{2.11}$$

Since variance gives us a sense of the spread between the scores, we use it to normalize them dividing by $\sqrt{d}$. Until now, we have the $n \times n$ matrix

$$\frac{QK^\top}{\sqrt{d}} \tag{2.12}$$

which has normalized coefficients and gives us an idea of how related is each input of the set $\mathcal{X}$ to the others. This normalization is desirable because it helps to keep calculations smooth, but is not sufficient. The next step is to transform the normalized scores to a probability distribution. For that, we use the Softmax function [2].

Definition 2.9. Let $z = (z_1, \ldots, z_m) \in \mathbb{R}^m$. The function $\sigma : \mathbb{R}^m \longrightarrow [0,1]^m$ defined by

$$\sigma(z)_i = \frac{\exp z_i}{\sum_{j=1}^{m} \exp z_j} \quad 1 \leq i, j \leq m \tag{2.13}$$

is called the **Softmax** function. [17]

If we apply the Softmax function to each row, we can interpret the scores defined before as probabilities drawn from a probability distribution for en element $x_q \in Q$ to observe an element $x_k \in K$. Essentially, the scores still preserve their objective, to gives us an idea of how related is each element x of $\mathcal{X}$ to the others. However, they are now in a more suitable formulation, stabilizing the training process and providing a non-linear re-weighting mechanism to concentrate the distribution of attention connections [18].

Up to this point, we have constructed a function that given the set $\mathcal{X}$, it gives us an estimation of how related are their elements among them:

$$\mathcal{S} = Softmax\left(\frac{QK^\top}{\sqrt{d}}\right) \tag{2.14}$$

where $Softmax$ refers to the the transformation (2.13) applied to each row of the matrix. Notice that this transformation ensures the positive semidefiniteness of our matrix $\mathcal{S}$, which avoids considering negatively-correlated information [2].

The next and final step is to use this information to extract the features that we are interested in. This is done by multiplying the matrix V with (2.14).

[2] Softmax function is a popular choice for normalization, but there are alternatives. In fact, some authors argue that there are better options when used with attention. Check [16] for more details.

Definition 2.10. Given a set $\mathcal{X}$ and its Q, K and V matrices, we define the function

$$SelfAttention(Q, K, V) = Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{2.15}$$

The output of this function is an $n \times d$ matrix which contains the desired information. In Chpt. 4 we will explain how we use it to perform a specific task.

2.2.1 Computational complexity of self-attention

The computational cost of self attention is its main drawback as we shall see in this subsection, where we analyze the complexity of computing (2.15). Let's decompose the calculus of self attention step by step and analyze their computational cost.

- $QK^\top$: When computing each score via (2.9), we do d multiplications. There are n^2 scores, therefore the cost of this calculus is $\mathcal{O}(dn^2)$.
- $\frac{QK^\top}{\sqrt{d}}$: We divide n^2 elements by $\sqrt{d}$. Hence, $\mathcal{O}(n^2)$.
- $Softmax(\frac{QK^\top}{\sqrt{d}})$: Applying (2.13) implies doing the exponentiation of n^2 numbers and afterwards n sums, which their results are used to do n^2 division. Therefore, $\mathcal{O}(n + 2n^2)$.
- $Softmax(\frac{QK^\top}{\sqrt{d}})V$: We do a matrix multiplication of a $n \times n$ matrix with $n \times d$. Hence, $\mathcal{O}(n^2 d)$

Adding up all steps cost, we achieve a complexity of

$$\mathcal{O}\left((2d + 3)n^2 + n\right) \tag{2.16}$$

In the vast majority of tasks where self attention is applied, n tends to be very large compared to d. Therefore, we obtain a complexity of $\mathcal{O}(n^2)$. Recall that n is the number of elements of the set $\mathcal{X}$ that we want to analyze, implying that the cost of the self-attention scales quadratically with the large of the set. As we have commented earlier, self attention was designed for NLP tasks. It is easy to consider inputs texts with a large number of words. As an example, this work has more than 9000 words, being 9000^2 of the order of 10^7. That is a huge drawback, since this method will be prohibitive when we are dealing with large inputs.

Moreover, the memory to store $M_{n\times n}$ matrices also becomes cumbersome with large values of n. In Chpt. 5 we shall see that is common practice to compute (2.15) in parallel during training. Hence, it becomes problematic to store all self-attention matrices when n is large.

In the following section, we present some approaches to overcome this issue.

Chapter 3

Efficient transformers

As seen in Sect. 2.2.1, the $\mathcal{O}(n^2)$ complexity is a main limitation of self attention. There is a general interest on reducing this complexity to a lower order of magnitude regarding n, allowing larger inputs to be handled. Commonly known as **linear attention mechanisms**, different methods have been presented to achieve a computational cost of $\mathcal{O}(n)$, where the quadratic complexity is suppressed. In this regard, in this project we refer to the full taxonomy [19] to divide these methods into three main categories. Within them, we can extract three main groups:

- **Sparse attention**: they focus on reducing the number of elements of the set $\mathcal{X}$ that each element attends to. In other words, it reduce the number of scores, reducing the dimension of $QK^\top$ based on different criteria derived from the task and input.
- **Attention reformulations**: their objective is to present computationally more efficient alternatives to $Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)V$ through the rearrangement of operators.
- **Low-rank methods**: they assume that $Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)$ contain redundancies and is enough to compute an approximated version, which is dimensionally smaller and thus reduce its computational cost.

Sparse attention mechanisms tend to be set dependant. The criteria used to decide how to restrict the scores is determined by empirical comparison among different methods and inputs, being out of scope of this work. Instead, we focus on attention reformulations and low-rank methods.

The first relies exclusively on mathematical formulation and a straightforward analysis can be derived. On the other hand, low-rank methods redundancies are also task depending. Nonetheless, a strong mathematical reformulation to find approximation matrices is required, and we focus on describing this process. More-

over, some of the most representative and popular implementations of attention reformulation and low-rank methods are carefully described. During all this section, we use Chpt. 2 notation and refer to $Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)$ as $\mathcal{S}$.

3.1 Attention reformulations

The idea behind these methods is to exploit mathematical properties of both operators and matrices to rearrange the formulation of $\mathcal{S}V$ into a more efficient calculation. The most common approach, denoted as *right-to-left trick*, is to compute first $K^\top V$ instead of $QK^\top$. Since $K^\top \in M_{d\times n}$ and $V \in M_{n\times d}$, the complexity is $\mathcal{O}(nd^2)$ instead of $\mathcal{O}(n^2 d)$, achieving a linear complexity on n.

However, notice that with this method Softmax normalization can not be applied. We are discarding the main idea behind our construction, the calculation of $\mathcal{S}$ matrix (2.14), which gave us the correlation among the inputs. Moreover, we are losing the normalization of our scores, losing the positive semidefinite property and the probabilistic approach. Therefore, it is necessary to reformulate the kernel function providing this information, achieving a substitute probability scores matrix $\mathcal{S}'$ for $\mathcal{S}$. Concretely, a new decomposable formulation is the objective. Suppose that ϕ is a function

$$\phi : M_{n\times d} \longrightarrow M_{n\times d} \tag{3.1}$$

that modifies Q and K in a way that a new matrix $\mathcal{S}'$ can be expressed as

$$\mathcal{S}' = \phi(Q)\phi(K^\top) \tag{3.2}$$

Then we are allowed to compute first the product $\phi(K^\top)V$, reducing the self attention complexity

$$SelfAttention(Q,K,V) = \mathcal{S}'V = \left(\phi(Q)\phi(K^\top)\right)V = \phi(Q)\left(\phi(K^\top)V\right) \tag{3.3}$$

The formulation of alternative scores matrix using a function ϕ is the main topic of different recent transformers, where different proposes are done in order to obtain a scores matrix allowing to compute self attention as explained at Sect. 3.3.

3.1.1 Cosformer

The Cosformer method [2] is based on the aforementioned approach, presenting a reformulation still based on dot-product but avoiding the use of the Softmax function. Instead, it utilizes a ReLU function,

Definition 3.1. *A ReLU is a function defined as*

$$\begin{aligned} ReLU : \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto max(0, x) \end{aligned}$$

We understand the application of the ReLU function to a matrix as applying the function to all its coefficients.

First, the ReLU function is applied to Q and K to ensure positive semidefiniteness of our approximation matrix $\mathcal{S}'$

$$Q' = \text{ReLU}(Q) \quad K' = \text{ReLU}(K) \tag{3.4}$$

The dot product is computed among Q' and K', obtaining the scores matrix $Q'K'^{\top}$. Next, a row wise normalization is applied instead of the Softmax normalization, being the rows of $\mathcal{S}'$ defined as

$$\mathcal{S'}_i = \frac{\sum_{j=1}^{n} \left(Q'_i K'^{\top}_j \right) V_j}{\sum_{j=1}^{n} \left(Q'_i K'^{\top}_j \right)} \quad 1 \leq i \leq n \tag{3.5}$$

Since we have avoided the Softmax, we can derive a decomposable formulation. To do so, Cosformer adds weights based on the cosinus to the coefficients of $Q'_i K'^{\top}_j$ and applies the Ptolemy's theorem [20] to obtain the desired formulation of $\mathcal{S}'$. The modified coefficients are expressed as

$$\mathcal{S'}_{ij} = Q'_i K'^{\top}_j \cos\left(\frac{\pi}{2} \times \frac{i-j}{\alpha} \right) \tag{3.6}$$

where α is a constant.

Applying Ptolemy's theorem:

$$\begin{aligned} Q'_i K'^{\top}_j \cos\left(\frac{\pi}{2} \times \frac{i-j}{\alpha} \right) &= Q'_i K'^{\top}_j \left[\cos\left(\frac{\pi i}{2\alpha} \right) \cos\left(\frac{\pi j}{2\alpha} \right) + \sin\left(\frac{\pi i}{2\alpha} \right) \sin\left(\frac{\pi j}{2\alpha} \right) \right] \\ &= \left(Q'_i \cos\left(\frac{\pi i}{2\alpha} \right) \right) \left(K'_j \cos\left(\frac{\pi j}{2\alpha} \right) \right)^{\top} + \left(Q'_i \sin\left(\frac{\pi i}{2\alpha} \right) \right) \left(K'_j \sin\left(\frac{\pi j}{2\alpha} \right) \right)^{\top} \\ &= Q_i^{cos} K_j^{cos} + Q_i^{sin} K_j^{sin} \end{aligned} \tag{3.7}$$

Therefore, we obtain our final $\mathcal{S}'$

$$\mathcal{S}' = Q^{cos} K^{cos} + Q^{sin} K^{sin} \tag{3.8}$$

with Prop. 2.6 ensuring its positive semidefiniteness. We can compute self attention using it:

$$\begin{aligned} SelfAttention(Q, K, V) = \mathcal{S}'V &= \left(Q^{cos} K^{cos} + Q^{sin} K^{sin} \right) \\ &= Q^{cos} \left(K^{cos} V \right) + Q^{sin} \left(K^{sin} V \right) \end{aligned} \tag{3.9}$$

With this approach, we avoid computing the $QK^\top$ product. Instead, we compute $K^{cos}V$ and $K^{sin}V$ first. Hence, we computed the following steps with associated complexity:

- Calculate $2n \cos\left(\frac{\pi i}{2\alpha}\right)$ and $2n \sin\left(\frac{\pi i}{2\alpha}\right)$. Therefore, $\mathcal{O}(4n)$
- Product of Q and K rows coefficients with the sinus and cosinus. $\mathcal{O}(2nd)$
- K^{cos} and K^{sin} product with V. $\mathcal{O}(2nd^2)$
- $Q^{cos}\left(K^{cos}V\right)$ and $Q^{sin}\left(K^{sin}V\right)$. $\mathcal{O}\left(2nd^2\right)$

Summing up all steps, its total complexity is $\mathcal{O}(4n + 2nd + 4nd^2)$. Therefore, considering $d << n$ as stated in Sect. 2.2.1, a computational cost of $\mathcal{O}(n)$ is achieved.

3.2 Low-rank

Low-rank methods, as their name indicate, argue that $\mathcal{S}$ is low-rank. This can be understood as data containing redundancies and that a reduced form of $\mathcal{S}$ is enough to represent the meaningful scores information. If we are able to generate a low-rank approximation matrix of $\mathcal{S}$, $\mathcal{S}' \in M_{k\times d}$ with $k < n$, we may use it to efficiently compute the $\mathcal{S}V$ product. Therefore, the computational cost of this matrix multiplication would be $\mathcal{O}(nkd)$ instead of the original $\mathcal{O}(n^2d)$ complexity. The existence of an approximation matrix is not trivial and needs to be proven. In this section, we present an statistical demonstration of such existence and two popular methods to build a low-rank approximation matrix: Nyströmformer [3] and Linformer [4].

3.2.1 $\mathcal{S}$ is low rank

In this subsection we present a theorem [4] ensuring that the probability for a low-rank approximation to exist is positive and converges to 1.

In order to do so, we first present two preliminary lemmas needed to proof the final result. The first lemma states:

Lemma 3.2. *[21] Let X be a probability drawn from a normal distribution $N(0, \sigma)$. Then, for any $\alpha < \frac{1}{2\sigma^2}$.*

$$\mathbb{E}[\exp(\alpha X^2)] = \frac{1}{\sqrt{1 - 2\alpha\sigma^2}} \tag{3.10}$$

Proof.

$$\begin{aligned}
\mathbb{E}[\exp(\alpha X^2)] &= \int_{\infty}^{-\infty} \exp(\alpha x^2)\, N(0,\sigma) dx \\
&= \int_{\infty}^{-\infty} \exp(\alpha x^2)\, \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\
&= \int_{\infty}^{-\infty} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}(1-2\alpha\sigma^2)\right) dx \\
&= \frac{1}{\sqrt{1-2\pi\sigma}} \int_{\infty}^{-\infty} \frac{\sqrt{1-2\pi\sigma}}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}(1-2\alpha\sigma^2)\right) dx
\end{aligned} \tag{3.11}$$

Using that

$$\int_{\infty}^{-\infty} \frac{\sqrt{1-2\pi\sigma}}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}(1-2\alpha\sigma^2)\right) dx = \int_{\infty}^{-\infty} N(0, \sigma/\sqrt{1-2\pi\sigma}) = 1 \tag{3.12}$$

We obtain our final result (3.10). □

This result is just a relationship that allow us to proof the following lemma, which is the core of the final theorem proof. Being a version of the Johnson - Lindenstrauss lemma [22], it states:

Lemma 3.3. *[21] Let $R \in M_{k\times n}$, $1 \leq k \leq n$, with independent and identically distributed entries from a normal distribution $N(0, 1/k)$. For any $x, y \in \mathbb{R}^n$, any $\epsilon > 0$, we have*

$$\begin{aligned}
\Pr(\|Rx\| \leq (1+\epsilon)\|x\|) &> 1 - \exp\left(-(\epsilon^2-\epsilon^3)\frac{k}{4}\right), \\
\Pr\left(\|xR^\top Ry^\top - xy^\top\| \leq \epsilon\|xy\|\right) &> 1 - 2\exp\left(-(\epsilon^2-\epsilon^3)\frac{k}{4}\right)
\end{aligned} \tag{3.13}$$

where Pr *denotes the probability and* $\|\cdot\|$ *the Euclidean norm.*

Proof. We proof the first inequality of the theorem. The second one is derived from an analogous procedure.

$$\begin{aligned}
\Pr(\|Rx\| \leq (1+\epsilon)\|x\|) &= \Pr(\|R\| \leq (1+\epsilon)) \\
&= \Pr(\exp(\|R\|) \leq \exp(1+\epsilon)) \\
\text{(a)} &\leq \frac{\mathbb{E}[\exp(\alpha\|R\|)]}{\exp(1+\epsilon)} \\
&= \frac{\prod_{j=1}^{k} \mathbb{E}\left[\exp\left(\alpha R_j^2\right)\right]}{\exp(1+\epsilon)} \\
\text{(b)} &= \left(\frac{\mathbb{E}\left[\exp\left(\alpha R_1^2\right)\right]}{\exp(1+\epsilon)}\right)^k
\end{aligned} \tag{3.14}$$

where at (a) we have used Markov's inequality[1] and at (b) the independence of the R rows. Now, for any $\alpha < \frac{k^2}{2}$, using Lemma 3.10 we have

$$\Pr\left(\|Rx\| \leq (1+\epsilon)\|x\|\right) \leq \left(\frac{\exp(-2(1+\epsilon)\alpha)}{\sqrt{1-2\alpha}}\right)^{\frac{k}{2}} \tag{3.15}$$

Selecting $\alpha = \epsilon/2(1+\epsilon)$,

$$\begin{aligned}\Pr\left(\|Rx\| \leq (1+\epsilon)\|x\|\right) &\leq \left((1+\epsilon)\exp(-\epsilon)\right)^{\frac{k}{2}} \\ &\leq \exp\left(-(\epsilon^2-\epsilon^3)\frac{k}{4}\right)\end{aligned} \tag{3.16}$$

and we obtain our result. □

With this result we have all necessary tools to present the main theorem of this section.

Theorem 3.4. *[4] For any Q, K, $V \in M_{n\times d}$, for any column vector $v \in \mathbb{R}^n$ of V, there exists a low-rank matrix $\tilde{\mathcal{S}} \in M_{n\times n}$ such that*

$$\Pr\left(\|\tilde{\mathcal{S}}v^\top - \mathcal{S}v^\top\| < \epsilon\|\mathcal{S}v^\top\|\right) > 1 - o(1) \tag{3.17}$$

and $rank(\tilde{\mathcal{S}}) = \Theta(\log n)$.

Proof. Let's denote $A = \frac{QK^\top}{\sqrt{d}} \in M_{n\times n}$ and a_{ij} its coefficients. We can reformulate $\mathcal{S}$ as

$$\exp(A) \cdot D_A^{-1} \tag{3.18}$$

where D_A^{-1} is a $M_{n\times n}$ diagonal matrix satisfying

$$(d_A)_{ii} = \sum_{j=1}^{n} \exp a_{ij} \tag{3.19}$$

and $\exp(A)$ is the matrix with coefficients $\exp(a_{ij})$.

Consider $R \in M_{n\times k}$ a matrix with independent and identically distributed entries from a normal distribution $N(0, 1/k)$. We define

$$\tilde{A} = \exp(A) \cdot D_A^{-1} R^\top R \tag{3.20}$$

whose rank satisfies that

$$rank(\tilde{A}) \leq rank(R) = k \tag{3.21}$$

[1] If X is a random variable and $a > 0$, then $\Pr(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$

Now select any $\epsilon > 0$, any row vector $A_i \in \mathbb{R}^n$ of A and any column vector $v \in \mathbb{R}^n$ of V. Applying Lemma 3.3,

$$\Pr\left(\|A_i R^\top R v^\top - A_i v^\top\| \leq \epsilon \|A_i v^\top\|\right) > 1 - 2\exp\left(-(\epsilon^2 - \epsilon^3)\frac{k}{4}\right) \tag{3.22}$$

Hence, we have

$$\begin{aligned}\Pr\left(\|\tilde{A} v^\top - A v^\top\| \leq \epsilon \|A v^\top\|\right) &= \Pr\left(\|A R^\top R v^\top - A v^\top\| \leq \epsilon \|A v^\top\|\right) \\ &\text{(a)} \geq 1 - \sum_i^n \Pr\left(\|A_i R^\top R v^\top\| < \epsilon \|A_i v^\top\|\right) \\ &\text{(b)} > 1 - 2n\exp\left(-(\epsilon^2 - \epsilon^3)\frac{k}{4}\right)\end{aligned} \tag{3.23}$$

Where in (a) we applied the union bond and at (b) Lemma 3.3. Selecting $k = 5\log(n)/(\epsilon^2 - \epsilon^3)$, the result is achieved. □

With this result, we proved the existence of a low-rank approximation matrix for $\mathcal{S}$. In conclusion, self attention is actually low-rank and we are allowed to look for $\mathcal{S}$ approximations. Nevertheless, there is no warranty that we can actually learn it, and one may fail to find it.

In the following parts, two popular low rank methods are presented.

3.2.2 Linformer

Linformer [4] is the first low-rank method that we describe. Its paper [4] was the first to present the above low-rank theorem and uses it to provide a straightforward approach to generate an approximation matrix $\mathcal{S}'$. The main idea behind the Linformer is to map the K and V matrices to a lower dimensional space and compute self-attention with their reduced dimension, therefore lowering the complexity. The following theorem warranties that this approach can be done.

Theorem 3.5. *For any $Q, K, V \in M_{n\times d}$, if $k = \min\{\Theta(9d\log(d)/\epsilon^2), 5\Theta(\log(n)/\epsilon^2)\}$, then there exists matrices $E, F \in M_{k\times n}$, $k < n$, such that, for any row vector w of matrix $\mathcal{S}$, we have*

$$\Pr\left(\|\sigma\left(wE^\top\right) FV - \sigma(w)V'\| \leq \epsilon \|\sigma(w)\|\|V\|\right) > 1 - o(1) \tag{3.24}$$

where σ refers to the softmax operation defined at (2.13).

Proof. Proof at [4]. □

Now we describe how the Linformer constructs $\mathcal{S}'$. Let E and F be two $M_{k\times n}$ learnable matrices, $k < d$, and K', V' a reduced version of K and V defined by

$$K' = EK \quad V' = FV, \quad K', V' \in M_{k\times d} \tag{3.25}$$

Linformer uses K' instead of K to compute a reduced version of $\mathcal{S}$,

$$\mathcal{S}' = Softmax\left(\frac{QK'^{\top}}{\sqrt{d}}\right) \tag{3.26}$$

which lives at $M_{n\times k}$ instead of $M_{n\times d}$. Afterwards, it applies the $\mathcal{S}V$ product using $\mathcal{S}'$ and V', obtaining

$$SelfAttention(Q, K, V) = \mathcal{S}'V' \tag{3.27}$$

If we analyze the computational cost of this approach, we have:

- Computation of K' and V'. $\mathcal{O}(2nkd)$,
- Apply Softmax. In this case, it implies doing the exponentiation of nk coefficients and afterwards k sums, which their results are used to do nk divisions. Hence $\mathcal{O}(k + 2nk)$.
- $\mathcal{S}'V$ product. $\mathcal{O}(ndk)$.

Summing up all steps, we obtain a complexity of $\mathcal{O}(3ndk + 3nk + k)$. Considering $d << n$ as usual, and also selecting a k such that $k << n$, the computational cost of the linformer is $\mathcal{O}(n)$.

3.2.3 Nyströmformer

The second low rank method that we present is based on the Nyström approximation method, a common technique to find low rank approximations of a matrix from a subset of its columns [23, 24]. First, we illustrate how this method can be applied to our matrix $\mathcal{S}$ and discuss why a direct implementation does not work for our situation. Afterwards, we describe the Nyström base model presented by Xiong [3].

Consider that $\mathcal{S} \in M_{n\times n}$ can be expressed in the following way,

$$\mathcal{S} = \begin{pmatrix} A_s & B_s \\ F_s & C_s \end{pmatrix} \tag{3.28}$$

where $A_s \in M_{m\times m}$, $B_s \in M_{m\times(n-m)}$, $F_s \in M_{(n-m)\times m}$ and $C_s \in M_{(n-m)\times(n-m)}$, with $m < n$. The submatrix A_s is called the *sample* matrix and is the one used to derive

an approximation matrix. Computing the singular value decomposition (SVD) of A_s

$$A_s = U\Lambda V^\top \quad U, V, \Lambda \in M_{m\times m} \tag{3.29}$$

where U, V are orthogonal and Λ is diagonal, Xiong defined the Nyström form of $\mathcal{S}$ as

$$\tilde{\mathcal{S}} = \begin{pmatrix} A_s \\ F_s \end{pmatrix} A_s^+ \begin{pmatrix} A_s & B_s \end{pmatrix} \tag{3.30}$$

In the last equation, A_s^+ refers to the Moore-Penrose inverse of A_s, i.e. it satisfies:

- $A_s A_s^+ A_s = A_s$
- $A_s^+ A_s A_s^+ = A_s^+$
- $(A_s A_s^+)^\top = A_s A_s^+$ and $(A_s^+ A_s)^\top = A_s^+ A_s$

Now, let's see how the coefficients of $\tilde{S}$ can be computed. Consider a row vector of Q, Q_i, and a row vector of K, K_j. One defines

$$\psi_K(Q_i) = Softmax\left(\frac{Q_i K^\top}{\sqrt{d}}\right) \quad \psi_Q(K_j) = Softmax\left(\frac{QK_j^\top}{\sqrt{d}}\right) \tag{3.31}$$

where $\psi_K(Q_i), \psi_Q(K_j) \in \mathbb{R}^n$. This step is equivalent to selecting the columns and rows of $\mathcal{S}$ respectively. Selecting m rows and m columns from them, denoted as $[\cdot]_{m\times 1}$, we construct

$$\begin{aligned} \phi_K(Q_i) &= \Lambda^{-\frac{1}{2}} V^\top \left[\psi_K^\top(Q_i)\right]_{m\times 1} \\ \phi_Q(K_j) &= \Lambda^{-\frac{1}{2}} U^\top \left[\psi_Q(K_j)\right]_{m\times 1} \end{aligned} \tag{3.32}$$

With $\phi_K(Q_i)$ and $\phi_Q(K_j)$ we compute the coefficients of $\tilde{\mathcal{S}}$

$$\tilde{\mathcal{S}} = \phi_K^\top(Q_i)\phi_Q(K_j) \quad 1 \le i, j \le n \tag{3.33}$$

Expressed as a matrix equation,

$$\tilde{\mathcal{S}} = \left[Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)\right]_{n\times m} A_s^+ \left[Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)\right]_{m\times n} \tag{3.34}$$

where $[\cdot]_{m\times n}$ notation means taking the m rows with n column values as before. If we compare with equation (3.30),

$$\begin{pmatrix} A_s \\ F_s \end{pmatrix} = \left[Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)\right]_{n\times m} \quad \begin{pmatrix} A_s & B_s \end{pmatrix} = \left[Softmax\left(\frac{QK^\top}{\sqrt{d}}\right)\right]_{m\times n} \tag{3.35}$$

At this point, one can multiply first the $m \times n$ matrix with V, as we explained at Sect. 3.1 with the right-to-left trick. Doing so, the complexity of the product

$\mathcal{S}V$ is reduced to $\mathcal{O}(nmd + m^2d + nmd)$, each complexity term corresponding to each matrix multiplication of (3.34). However, the method described before is not effective for our case since it does need of the prior computation of $\mathcal{S}$. Therefore, as we have seen in Sect. 2.2.1, the $\mathcal{O}(n^2)$ is still present.

Xiong modified the latter process in order to avoid the $\mathcal{S}$ computation beforehand. First, it defines the concept of *landmark*.

Definition 3.6. For any matrix $M \in M_{n\times d}$, any set of m vectors in $\mathbb{R}^d$, $m < n$, derived from the rows of M is named a **landmark** of M. The matrix generated from this set of vectors is called the **landmark matrix**.

The main idea of the Nystömformer is to use Q and K landmarks to first generate the A_s matrix and next compute the approximations.

Suppose we have selected a pair of landmarks from Q and K and created with them their landmarks matrices, denoted as $\tilde{Q}$ and $\tilde{K}$. We can produce the matrix A_s as

$$A_s = Softmax\left(\frac{\tilde{Q}\tilde{K}^\top}{\sqrt{d}}\right) \tag{3.36}$$

which accepts a SVD decomposition as (3.29).

Following an analogous process as before, we define

$$\psi_{\tilde{K}}(Q_i) = Softmax\left(\frac{Q_i\tilde{K}^\top}{\sqrt{d}}\right) \quad \psi_{\tilde{Q}}(K_j) = Softmax\left(\frac{\tilde{Q}K_j^\top}{\sqrt{d}}\right) \tag{3.37}$$

and from them,

$$\begin{aligned} \phi_{\tilde{K}}(Q_i) &= \Lambda^{-\frac{1}{2}}V^\top\left[\psi_{\tilde{K}^\top}(Q_i)\right]_{m\times 1} \\ \phi_{\tilde{Q}}(K_j) &= \Lambda^{-\frac{1}{2}}U^\top\left[\psi_{\tilde{Q}}(K_j)\right]_{m\times 1} \end{aligned} \tag{3.38}$$

which allow us to compute the coefficients of an approximated matrix

$$\mathcal{S}'_{ij} = \phi_{\tilde{K}}^\top(Q_i)\phi_{\tilde{Q}}(K_j) \quad 1 \le i,j \le n \tag{3.39}$$

Finally, we obtain our approximation matrix:

$$\mathcal{S}' = \left(Softmax\left(\tfrac{Q\tilde{K}^\top}{\sqrt{d}}\right)\right)\left(Softmax\left(\tfrac{\tilde{Q}\tilde{K}^\top}{\sqrt{d}}\right)\right)^+\left(Softmax\left(\tfrac{\tilde{Q}K^\top}{\sqrt{d}}\right)\right) \tag{3.40}$$

Therefore, we can now compute the $\mathcal{S}'V$ in a less expensive way, as we previously stated. However we still have to check if the $\mathcal{O}(n^2)$ is avoided. To check this, we must answer the following questions:

1. Which is the computational cost of the SVD of A_s?
2. How do we select the landmarks?

The first is answered by the following lemma.

Lemma 3.7. *[25] For $A_s \in \mathbb{R}^{m\times m}$, the sequence $\{Z_j\}_{j=0}^{j=\infty}$ generated by*

$$Z_{j+1} = \frac{1}{4} Z_j(13I - A_s Z_j(15I - A_s Z_j(7I - A_s Z_j))) \tag{3.41}$$

converges to the Moore-Penrose inverse A_s^+ in the third order with initial approximation Z_0 satisfying $\|A_s A_s^+ - A_s Z_o\| < 1$.

Using the lemma, for each iteration the most expensive operation is the multiplication of $m \times m$ matrices. Considering that around 100 iterations are enough to converge [3], the Moore-Penrose inverse computation has a complexity of $\mathcal{O}(m^3)$.

Answering the second question, to compute the landmarks, the following formulas derived from a NLP approach to compute local average pooling [26] are presented:

$$\tilde{Q}_j = \sum_{i=(j-1)\times l+1}^{(j-1)\times l+m} \frac{Q_i}{m} \qquad \tilde{K}_j = \sum_{i=(j-1)\times l+1}^{(j-1)\times l+m} \frac{K_i}{m} \quad 1 \leq j \leq m \tag{3.42}$$

where $l = n/m$ and M_i denotes the row of a matrix as usual. Using (3.42), we are extracting m rows form Q and computing an average sum among them to produce each row of the landmark matrix. For each row, we compute m sums. For each sum, we do d divisions, since $Q_i, K_i \in \mathbb{R}^d$. Finally, the number of rows of $\tilde{Q}$ and $\tilde{K}$ is n. Hence, since $d << n$ and assuming $m << n$, the computational cost of (3.42) is $\mathcal{O}(n)$.

In conclusion, with all the steps considered, we have:

- Landmarks computation. $\mathcal{O}(2n)$
- The 3 softmax of equation (3.40). With the same argument done previously with other methods, they have a complexity of $\mathcal{O}(m + 2nm)$ for the $M_{n\times m}$ and $M_{m\times n}$ matrices, and $\mathcal{O}(m + 2m^2)$ for the $M_{m\times m}$ matrix.
- Moore-Penrose inverse of $Softmax\left(\frac{\tilde{Q}K^\top}{\sqrt{d}}\right)$. $\mathcal{O}(m^3)$
- Multiplication of $Softmax\left(\frac{\tilde{Q}K^\top}{\sqrt{d}}\right)$ with V. $\mathcal{O}(nmd)$
- Rest of the matrix multiplications. $\mathcal{O}(m^2 d + m^2 n)$

Adding up all the steps, and , the Nystromformer method has a complexity of

$$\begin{aligned} &\mathcal{O}\left(2n + 3m + 4nm + 2m^2 + m^3 + nmd + m^2 d + m^2 n\right) = \\ &\mathcal{O}\left(n(2 + 4m + md + m^2) + 3m + m^2(2 + d) + m^3\right) \end{aligned} \tag{3.43}$$

Assuming $d << n$ and $m << n$, we obtain a complexity of $\mathcal{O}(n)$.

Chapter 4

Transformer architecture

In Sect. 2 we described in great detail the self-attention mechanism. In this chapter we focus on presenting our Transformer model, inspired by [1, 27, 12], to exploit the self-attention mechanism to perform video classification.

During all this section, we use repeatedly linear projections of matrices, which are defined as follows:

Definition 4.1. A linear projection of a matrix $M \in M_{n\times m_1}$ into a space $M_{n\times m_2}$ is defined as applying a function

$$\begin{aligned} f : M_{n\times m_1} &\longrightarrow M_{n\times m_2} \\ M &\longmapsto M \cdot W + B \end{aligned}$$

where $W \in M_{m_1\times m_2}$ and $B \in M_{n\times m_2}$. W is called the **projection matrix** and B the **bias**.

If not specified, the bias term will be considered as the zero matrix and ignored. The coefficients of all the matrices W and B used in the transformer are called the **parameters** of the model.

4.1 Input embedding

Assume that our input are videos of $H \times W$ resolution, RGB encoding and with a total of T frames, which can be represented as a vector $x \in \mathbb{R}^{3\times H\times W\times T}$. The set of this videos is the set $\mathcal{X}$ of Sect. 2, and the first step is to decide a feature map which maps them to an Euclidean space where we can apply the dot product. The elements generated with the feature map are called the **tokens** of the model.

Due to their nature, video classification can be understood as performing two different tasks simultaneously:

1. Understand the spatial relationships of the frames. In other words, learn image reasoning.

2. Understand the temporal relationship among frames.

To do so, we must generate the tokens in a way that ensures that we handle the two tasks.

The first task can be linked to an image classification task. Self attention has been used in computer vision with remarkable success [27], and the most straightforward way to attack this problem is to generate patches from the images as shown in Fig. 4.1a.

Each image is divided in patches, which are the set $\mathcal{X}$ that would be fed to the self attention mechanism. As illustrated in Chpt. 1, we can link the NLP approach to vision by treating each patch as the word of a sentence (the image). In this way, self attention provides information on which image patches are most correlated. This technique is called **spatial tokenization**.

Similarly, the second task can be also understood as an NLP task, associating now the words with the whole image and the sentence with all the video. With this approach, self attention can be applied in a direct way, as shown in Fig. 4.1b, where the set $\mathcal{X}$ is formed by all the frames of the videos. Using this method, self attention learns how each frame interacts with the others in a temporal sense. We name this method **temporal tokenization**.

Combining both approaches, the simplest tokenization method to consider temporal and spatial dimensions at the same time is achieved, as proposed by [12]. Each frame is divided in patches and afterwards these patches are considered as the frames of the video and temporal tokenization is made, as illustrated in Fig. 4.1c. With this, we force self attention to learn both spatial and temporal features of the video. This video tokenization is called **spatial-temporal tokenization**.

Now we formalize the procedure explained above. First, for each frame $x_t \in \mathbb{R}^{3\times H\times W}$, $1 \leq t \leq T$, we generate P patches of size $H_p \times W_p$, where $P = \frac{H}{H_P} \cdot \frac{W}{W_P}$.

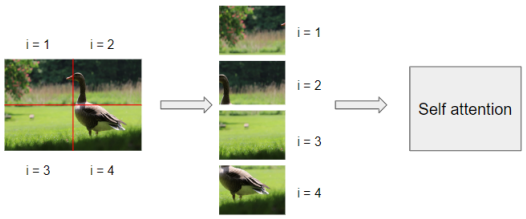
$$\begin{aligned} \phi_1 : \mathbb{R}^{3\times H\times W} &\longrightarrow P \times (\mathbb{R}^{3\times H_P\times W_P}) \\ x_t &\longmapsto (x_t^1, x_t^2, \ldots, x_t^P) \end{aligned}$$

Afterwards, we can project the patches to $\mathbb{R}^{3H_PW_P}$

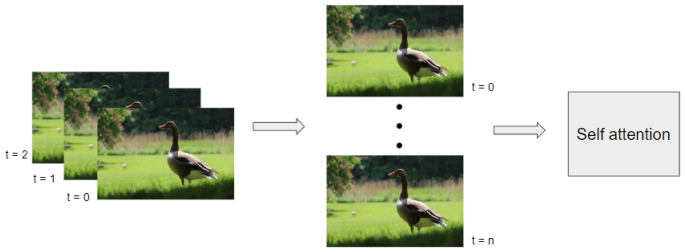
$$\begin{aligned} \phi_2 : P \times (\mathbb{R}^{3\times H_P\times W_P}) &\longrightarrow P \times \mathbb{R}^{3H_PW_P} \\ (x_t^1, x_t^2, \ldots, x_t^P) &\longmapsto (z_t^1, z_t^2, \ldots, z_t^P) \end{aligned}$$

Therefore, applying $\phi_2 \circ \phi_1$ to each video, we obtain $n = T \cdot P$ patches representations,

$$z_t^i \in \mathbb{R}^{3H_pW_p} \quad 1 \leq t \leq T, \quad 1 \leq i \leq P \tag{4.1}$$



(a) Spatial tokenization for an image



(b) Temporal tokenization for a video

(c) Spatial-temporal tokenization for a video

Figure 4.1: Tokenization methods described at Sect. 4.1.

that we denote per simplicity as

$$z_p, \quad 1 \leq p \leq n$$

where p denote the patch number. Notice that they are defined in a desired Euclidean space $\mathbb{R}^k$, with $k = 3H_P W_P$. With this, we have a suitable definition of our feature map, obtaining from our set what we call **patch embeddings**, which are the tokens of our model. Their dimension is named the **embedding dimension**.

4.1.1 Positional encoding

The previous formulation generates the elements of our set $\mathcal{X}$. However, by construction, self attention is positional invariant given the inherent properties

of the Euclidean product. For example, a video of a glass falling to the ground played backwards would be seen as a video of a glass being reconstructed. Hence, we must encode the positional information of each token in some way in order to fully comprehend what is happening on the video. The most common way is to slightly modify each token adding up a vector of the same dimension which contains this information. We denote the modifier vector as **positional encoding**. There are different ways to construct it, one of the most usual ones being using a sinusoidal function, defined by [1] as follows:

$$PE_{(p,2i)} = sin(p/010000^{\frac{2i}{k}}) \quad PE_{(p,2i+1)} = cos(p/10000^{\frac{2i}{k}})$$

where $1 \leq i \leq k$ and $1 \leq p \leq n$.

These equations generates a vector of $\mathbb{R}^k$ for each token p, which its components are unique. This vector is called **sinusoidal positional encoding**. Adding this vector to each patch embedding (4.1)

$$x_p = z_p + PE_{(p)}$$

we obtain our final input. The set $\mathcal{X}$ to which we will apply self attention is formed by the elements x_p, with $1 \leq p \leq n$.

4.2 Attention Block

The next step is to apply self-attention. Suppose that we have computed the $n \times d$ matrices Q, K and V as described in Sect. 2.2.

4.2.1 Multi-Head Attention

Usually, instead of performing a single attention step, it is common to project the query, key and values matrices into smaller dimensions and perform multiple times the self attention operation in parallel.

The intuition behind this is to allow the model to learn different properties of the input. For instance, dealing with a summarizing NLP task, each head can focus detecting different information, such as who is doing a certain action, where it is happening, etc. Every parallel computation of self attention is called a **head**.

Suppose we have a fixed number of heads H. We *split* the matrices H times. This means applying linear projections to generate $n \times d/H$ matrices

$$\begin{array}{ccc} f_i : M_{n\times d} \longrightarrow M_{n\times(d/H)} & g_i : M_{n\times d} \longrightarrow M_{n\times(d/H)} & h_i : M_{n\times d} \longrightarrow M_{n\times(d/H)} \\ Q \longmapsto Q^i & K \longmapsto K^i & V \longmapsto V^i \end{array}$$

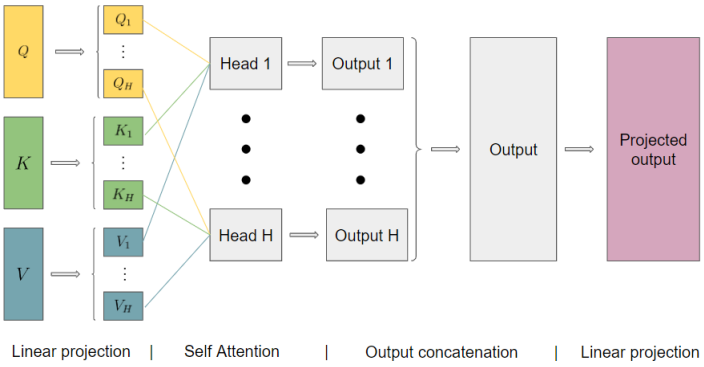


Figure 4.2: Multi-Head attention diagram

with $1 \leq i \leq H$. These linear projection are not fixed and their projection coefficients are learned by the model. The Q^i, K^i and V^i matrices are applied to the self attention function (2.15) and produce H outputs of size $n \times d/H$, $Softmax\left(\frac{Q^i {K^i}^\top}{\sqrt{d}}\right) \in M_{n \times d/H}$. Afterwards they are concatenated to recover a single matrix and linear projection projection to $M_{n \times d}$ is applied, obtaining the same matrix size as the original (2.14).

$$\mathcal{S} = Softmax\left(\frac{QK^\top}{\sqrt{d}}\right) = \left(Softmax\left(\frac{Q^1 {K^1}^\top}{\sqrt{d}}\right) \quad \dots \quad Softmax\left(\frac{Q^H {K^H}^\top}{\sqrt{d}}\right)\right) W_O \tag{4.2}$$

The latter linear projection role is to mix the information from all attention heads into a single output.

4.2.2 Feed forward

After performing the self attention, we have to transform the information extracted with self attention to a desirable format. This is done using two linear projections, this time also computing the bias, together with a ReLU non-linearity, defined at Def. 3.1.

ReLUs are commonly used in Machine Learning since it has been empirically proven that they boost the performance of the model and reduce their convergence time [28]. Moreover, for transformer models ReLU has an even more important role: it assures that the transformer can be considered as an universal function approximator for sequence-to-sequence functions [29]. In other words, they warranty the success in performing NLP tasks given a suitable optimization process. Even though we are not dealing specifically with sequence-to-sequence, this result can be extrapolated to other tasks, and it is why we can assume that our model

can learn how to retrieve the desired information.

Therefore, we use them and transform the self attention matrix (4.2) computed before as follows

$$\mathcal{S}_{out} = ReLU(0, \mathcal{S} \cdot W_1 + B_1) \cdot W_2 + B_2$$

where here the ReLU is applied to each element of S individually. The projections are chosen so that $\mathcal{S}_{out}$ lives in $M_{n\times k}$. Hence, each row of $\mathcal{S}_{out}$ can be interpreted as vector $x \in \mathbb{R}^k$, which is the space where our self attention's previous input lived. In this way, we can repeat the process described at Sect. 4.2.1, and compute self attention again. The number of times that one repeats the attention block is called the **depth** of the model.

4.2.3 Linear Classifier

After applying the attention blocks, the model has extracted all the information from the input and classification is the next and final step. Suppose that we have C different classes. To classify them, a simple linear projection to $M_{n\times C}$ is applied. Basically, the model is forced to produce an output for each class and for each element $x \in \mathcal{X}$. Afterwards, a Softmax is applied, giving us the probability distribution of each element x is of each class. Finally, the highest probability will be the class decided by the model.

$$\begin{aligned} g : M_{n\times k} &\longrightarrow M_{n\times C} \longrightarrow \mathbb{R}^n \\ S_{out} &\longmapsto M_{scores} \longmapsto Softmax(M_{scores}) \end{aligned}$$

This yields to a model which can classify videos. A diagram of the full model is represented at Fig. 4.3.

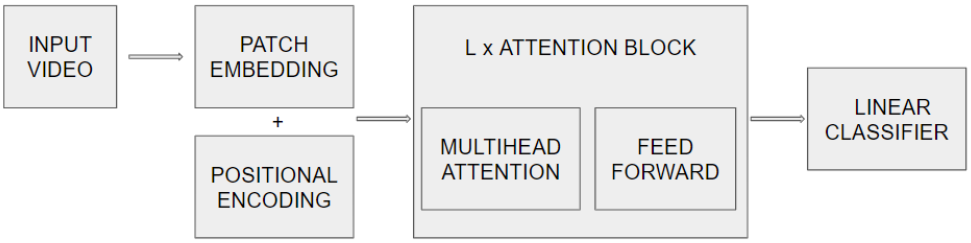


Figure 4.3: Our transformer architecture

Chapter 5

Experimental comparison between self-attention mechanisms

One of the main goals of our work is to compare how the different self attentions mechanisms described at Chpt. 3 perform on a video classification task. Specifically, our objective is to answer the following question: *Which self attention mechanism has the best performance/computational cost trade off for video classification?* To fulfill this objective, we must translate the mathematical formulation of our transformer architecture defined at Chpt. 4 to code and finetune its parameters, meaning that the coefficients of our linear projections are suited to classify video.

In order to do so, we follow the standard procedure for Machine Learning models based on backpropagation[1]. From a video dataset, we use three datasets to train and evaluate our model: the training, validation and test sets. We feed the model with a bunch of training set clips[2], named *batch*, and evaluate how good the model is performing with a loss function. Afterwards, a gradient descent method is applied to update the parameters. This process is repeated for the entire training set in an iterative way for a certain number of times, named *epochs*. After each epoch is completed, the model classifies the validation set clips to track how the model performs with unseen clips. The performance on the validation set will be analyzed by a selected metric, indicating us when the model has converged and the training is completed. Finally, once the training has finished, the model is evaluated using the training set to evaluate its performance. The hardware used to train our model is a Nvidia GeForce GTX 1080 Ti GPU.

In this section we present all elements related to the training, including both

[1]We assume that the main concepts of backpropagation and gradient descent are known by the reader. Check [30] for further detail.

[2]Notice how we use clips instead of videos. Performing this task with full videos become prohibitive due to their size. Hence, we use shorter clips derived from them.

the dataset generation and the hyperparameters choice, explaining the motivations behind them. Finally, we present our evaluation metrics and the results derived from them.

5.1 Dataset

The first step is to decide the data that we are going to use. We are interested in a dataset containing labelled videos, allowing us to classify them. Our choice have been the Epic-Kitchens-100 dataset [5], consisting of nearly 100 hours of first-person videos of daily kitchen activities.

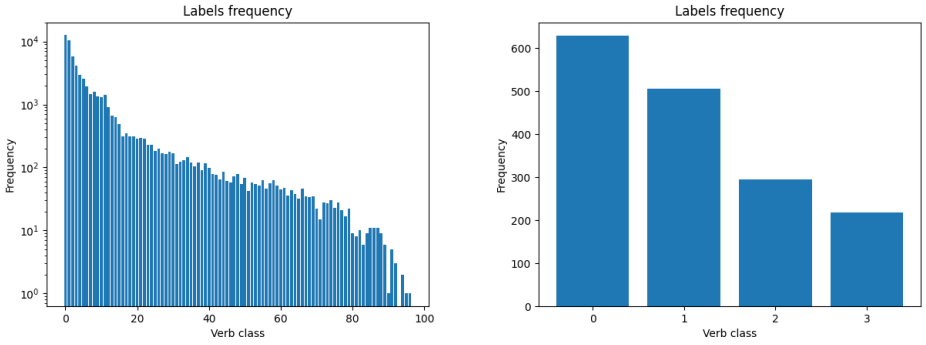
	Hours	Videos	Clips	Verb Classes	Noun Classes	Action classes
Train	74.7	495	67217	97	289	4053
Validation	13.2	138	9668	78	211	1352
Test	12.1	67	13092	84	207	1487

Table 5.1: Epic-Kitchens-100 Dataset

All video is disclosed in its frames, with an annotations file indicating the start and stop frame for each clip contained in the video, with its associated information. The dataset is conceived in a way that can be used for learning different tasks. For our concern, video classification, the dataset provides a verb, noun and action label associated with each clip. Since our main objective is to compare model performance, and not achieve state of the art results, we focus only on verb classification. In this way, the classification task becomes easier and the training is more straightforward.

The dataset is structured in the usual way as three main partitions: train set, validation set and test set. However, we decided to generate a custom dataset derived from the original one. There were two reasons that motivated this decision. Firstly, test partition was not publicly available and to inference a trained model it should be sent to the dataset owners. This procedure was not suitable for our case, since we wanted to inference several times different models. Secondly, the dataset was too big for our resources and time. We wanted to use 4 different self-attention mechanisms, and the time needed to train them with enough epochs would have been prohibitive. Therefore, we created custom train, validation and test sets in the following way.

To begin with, we reduced the number of classes to 4. These 4 classes represented almost a 60% of the whole dataset, and its selection was motivated since the original dataset was extremely unbalanced, as can be seen in Fig. 5.1a. If we wanted to learn the marginal classes we would have required a more exten-



(a) Original dataset labels frequency in log scale.

(b) Used dataset labels frequency.

Figure 5.1: Dataset labels.

sive training and powerful model, demanding time and resources that were not available.

From the 4 labels dataset, we randomly retrieved a 5% of the clips. Afterwards, we generated our train and validation set by splitting this subset in a 85/15 partition. In an analogous way, we selected the 4 main classes from the original validation set and extract a 5%. In this case, we did not apply any partition and this set served as our test set.

It may seem as we reduced our dataset too much. However, since we simplified the problem to only 4 classes, the models had an enough quantity of clips to learn, as we shall see in Sect. 5.3. The used dataset is described at Tab. 5.2.

	Hours	Videos	Clips	Verb Classes
Train	1.8	338	1648	4
Validation	0.3	168	291	4
Test	0.3	85	280	4

Table 5.2: Epic-Kitchens-100 modified dataset

5.1.1 Clips preprocessing

The dataset clips have a resolution of 1080×1920 and an average number of frames of 172. Feeding the model with the full clips becomes prohibitive since the GPU memory is not enough to handle such amount of information. Therefore, we crop the frames to 112×112 and fix the number of frames to 100. To fix the

number of frames we proceed as follows. Let T be the number of frames of a clip. First, we obtain a 200 frames clip. If $T < 200$, we apply zero padding, i.e. we add black frames to the end and beginning of the video. Otherwise, we randomly select a frame from the first $T - 200$ frames and extract the following 200 frames. Afterwards, we reduce the number of frames to the half by alternately selecting one frame out of two for every consecutive pair of frames. In this way, we ensure that we obtain enough temporal information.

5.2 Experimental Setup

We need a programming language and an experimental framework to build up our Transformer described at Chpt. 4. The selected programming language is Python and the specialized Machine Learning library PyTorch [13] is the selected framework, providing all necessary tools to build and train our model. Using them, we define all transformer modules and put them together to generate our model, as described at Fig. 4.3. Moreover, the efficient transformers formulations are also coded in a way that we can decide which mechanism the model uses. The three of them are available together with the naive implementation of self attention described at Chpt. 2.2, which from now on is referred to as Vanilla attention.

Nevertheless, two key architecture decisions remained to be made. *How many attention blocks L our model should contain and how many attention heads H each block should have?* More attention blocks and heads increase the capacity of our model to learn, but also demand more computations and memory. We want to find an optimal trade off between the batch size and L and H to not overload the memory and also ensure that the computations do not become prohibitive in terms of time. To select the best configuration, we performed a grid search with vanilla attention and the best result without overloading the memory was the next:

Clip resolution	Frames/clip	Batch size	Attentions heads	Attention Blocks
112×112	100	4	4	2

Table 5.3: Model and dataset final configuration

We also considered reducing the number of frames and their resolution, but lower of either of the two parameters was not enough for the model to learn.

Now that the model is defined, the next step is to define the loss function and the optimizer to fine-tune its parameters. The selected loss function is the **cross entropy loss**, which is suitable for multi label classification and is a standard choice for this scenario [31].

Definition 5.1. Let $x \in \mathbb{R}^C$ be the probabilities that the input is classified as the i-th class, for $1 \leq i \leq C$, and y the actual index class. Then, the **cross entropy loss** of this prediction is defined as

$$\sum_{i=1}^{C} -w_i \log \frac{\exp(x^i)}{\sum_{j=1}^{C} \exp(x^j)} \mathbb{1}_y(i) \tag{5.1}$$

where w_i is a weight associated to the i-th class and $\mathbb{1}_y(i)$ is the indicator function, being 1 if and only if $y = i$, otherwise being 0.

In our dataset, the verb classes are unbalanced, as we can appreciate at Fig. 5.1b. Therefore, we used specific w_i weights to handle this. If y is a label associated to m clips on the training set, n is the total number of training samples and C the number of classes, then its weight is computed as

$$w_y = \frac{n}{m \cdot C} \tag{5.2}$$

Notice how a class with little clips would have a small denominator, therefore influencing the loss with more strength than a common class. This increases the effect on the loss of minority classes and avoid the model to overfit to the main class, i.e. classify all classes to the most common one.

The next decision is to choose an optimizer to minimize the loss. Our selection was Adam [32], which is an algorithm for first-order gradient-based optimization that usually performs well in very large datasets and is suitable for high dimensional spaces [33]. The version used is described at Algo. 1.

Algorithm 1 Adam

Require: $\gamma, \beta_1, \beta_2, \theta_0$(parameters), $f(\theta)$(objective)
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  for $t = 1$ **to** ... **do**
    $g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$
    $m_t \leftarrow \beta_1 m_{t-1} + \left(1 - \beta_1\right) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + \left(1 - \beta_2\right) g_t^2$
    $\hat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right)$
    $\hat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right)$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon\right)$
  end for
  return θ_t

We used the default parameters for β_1, β_2 and empirically fine-tuned the learning rate, γ, by doing several brief trainings with Vanilla self attention mechanism.

We tried learning rates of the order of 10^{-8} to 10^{-3} and achieved the best result with a learning rate of 10^{-5}. Experimentation with learning rate scheduler was also realized but non significant improvements where appreciated.

Putting all together, we have our final training configuration:

Loss	**Optimizer**	γ	β_1	β_2
Cross Entropy Loss	Adam	10^{-5}	0.9	0.999

Table 5.4: Training configuration

5.3 Results

We trained our models for 50 epochs, evaluating the model with the validation set after each epoch as usual. We configured an early stopping, saving the model parameters which achieved the best metric. The results and all consequent conclusions derive from the early stopped models. The metric computed to evaluate the model performance during training was an average recall among the classification, being the recall defined as follows:

Definition 5.2. [34] Let t_p be the number of true positives and f_n the number of false negatives for a class. We define the **recall** of the class as

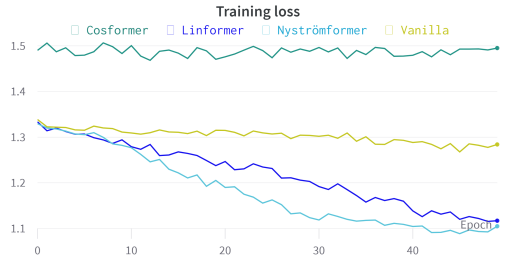
$$R = \frac{t_p}{(t_p + f_n)}$$

The obtained results for the training and validation set can be observed at Fig. 5.2, where the evolution of the loss and the average recall is shown. Some insights about our model can be derived from them.

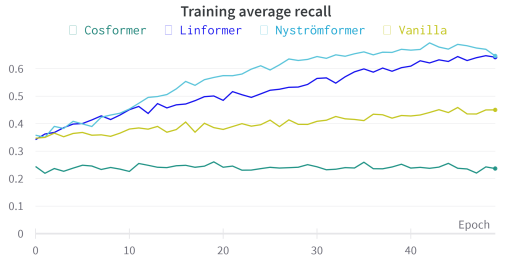
First, we directly appreciate from Fig. 5.2a and Fig. 5.2b that Cosformer fails to learn. The loss stays horizontal and the average recall stays around 25%, which is the same percentage a random classifier would provide. This can be motivated for two reasons.

On the one hand, it could be that the model and training configurations do not fit the Cosformer mechanism. We established a common framework, but the different self attention mechanisms performance can be highly influenced by the parameters used. In fact, the ideal situation would be to fine-tune the configuration for each of the mechanisms. The configuration fine-tuning, however, is costly and requires times, being out of scope for our work.

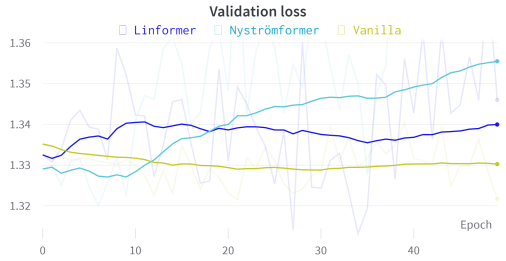
On the other hand, Cosformer authors empirically found that Cosformer can punish far-away connections and enforce locality [2]. This behaviour on our task



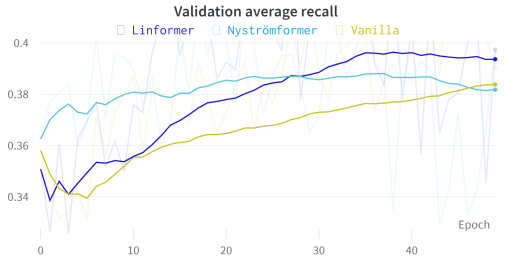
(a) Loss evolution for the training set



(b) Average recall evolution for the training set



(c) Loss evolution for the validation set



(d) Average recall evolution for the validation set

Figure 5.2: Metrics evolution during all training for the training and validation sets. For the validation set, we omitted the Cosformer results since the model failed to learn. For better visualization, an Exponential Moving Average smoothing was applied to the validation results.

might be critical, since the temporal dimension of videos requires of long term dependencies. Moreover, enforcing locality could be counterproductive. By how videos are tokenized, enforcing locality would imply paying attention to close patches. That could punish the temporal feature extraction, being detrimental to its performance.

On conclusion, from this results about Cosformer, one should first look for a more in depth training configuration, to check whether the negatives results are derived from a wrong set up. After this, an interesting path for further work would be to train larger models to actually inspect if enforcing locality on videos has a negative impact on their performance.

Now we analyse the behaviour of Nyströmformer and Linformer versus Vanilla, since both efficient self attention mechanisms succeed on training. At first glance, one can appreciate that Vanilla's metrics vary slowly, with both the loss and average recall evolving at low rate. On the contrary, Nyströmformer and Linformer loss for the training set decreases faster, consequently increasing the average re-

call also with a higher rate. However, the loss decreasing is not reflected at the validation set. For the Nyströmformer, we appreciate how the loss starts increasing on an early stage for the validation set, which indicates rapid overfitting of the training set. On the other hand, Linformer loss for the validation set remains almost horizontal. Nonetheless, for both mechanisms the average recall increases and achieve values similar to the Vanilla attention.

To fully comprehend and analyze how the models perform, we must use the test set to compare how well the models classify them. We first define some of the metrics used to evaluate our model.

Definition 5.3. The **accuracy** is defined as the number of correct predictions divided by the number of total samples.

Accuracy score does not take into account data unbalancing. For instance, with an unbalanced dataset with 80% of the class labels being 0, a model which assigned all labels to 0 would achieve a 80% accuracy, even though its capacity to solve the problem is clearly not good. In any case, accuracy is the most popular metric for classification tasks and we consider appropriate to present it.

To overcome accuracy weaknesses, we used the average of the recall defined at Def. 5.2, which was label sensitive and gave us a more suitable metric to monitor the training. Despite this, we would like further analysis for each class, to detect specifically how the model perform with each one. With this objective in mind, we use confusion matrices.

Definition 5.4. From a classification task of C classes, we define the **confusion matrix** as the $M_{C\times C}$ matrix with coefficients c_{ij} equal to the number of observations known to be in group i and predicted to be in group j.

Confusion matrices give us insights of which labels are more difficult to learn, and whether there is an over classification for a singular class. For a perfect model that classifies all clips perfectly, its confusion matrix would be a diagonal matrix with ones on its diagonal. The confusion matrices for our models are shown at Fig. 5.4.

Nonetheless, we need a metric which can quantify how good this predictions are, extracting the information from confusion matrices and that serves as a general indicator of the model performance. The metric used with this objective is the Matthews correlation coefficient.

Definition 5.5. [35] Let c_{ij} be the coefficients of a confusion matrix. Consider

- $t_k = \sum_i^K c_{ik}$ the number of times class k occurred
- $p_k = \sum_i^K c_{ki}$ the number of times class k was predicted

- $c = \sum_k^K c_{kk}$ the total number of samples correctly predicted
- $s = \sum_i^K c_{ij}$ the total number of samples

We define the **Matthews correlation coefficient** as

$$Matthews = \frac{c \cdot s - \sum_k^K p_k \cdot t_k}{\sqrt{\left(s^2 - \sum_k^K p_k^2\right) \cdot \left(s^2 - \sum_k^K t_k^2\right)}} \tag{5.3}$$

The Matthews coefficient lays in the range $[x, 1]$ with $-1 < x < 0$, being x determined by the number and distribution of ground true labels. A value of 1 indicates perfect prediction.

Using these metrics, being the Matthews coefficient our main criteria, and analyzing the computational cost of our models once trained, we can determine which model is the most efficient. The computational parameters computed are the number of floating points operations (FLOPs) performed inside the attention blocks during evaluation and the memory occupied while training. We compute only the FLOPs related to the attention blocks since they are the only variation inside our model when permuting the different self attention mechanisms. Regarding the memory, the one used during training is the analyzed in order to take into account both the memory dedicated to store the model parameters and their gradients. Therefore, computing the aforementioned metrics, we obtain the following results:

Model	Accuracy	Avg. recall	Matthews	Memory (MB)	FLOPs ($\times 10^9$)
Vanilla [1]	0.40	0.41	0.22	9707	148.05
Nyströmformer [3]	0.41	0.39	0.19	2215	0.98
Linformer [4]	0.41	0.34	0.16	2354	1.45
Cosformer [2]	0.16	0.25	0.0	3512	23.14

Table 5.5: Metrics derived from the test set.

We can plot the Matthews coefficient versus the FLOPs and perform an efficiency comparison, assigning to our models points a size proportional to the memory required. The obtained graphic can be observed at Fig. 5.2. The ideal efficient mechanism would be on the top left corner, achieving the highest performance metrics and the lowest number of FLOPs, and it would also have the smallest point, representing a minimal memory occupation. From this graphic, we observe how Nyströmformer achieves the best results, closely followed by Linformer. The graphic also empathizes how Vanilla achieves the highest Matthews coefficient but implies the highest cost, also demanding the largest amount of memory. Besides,

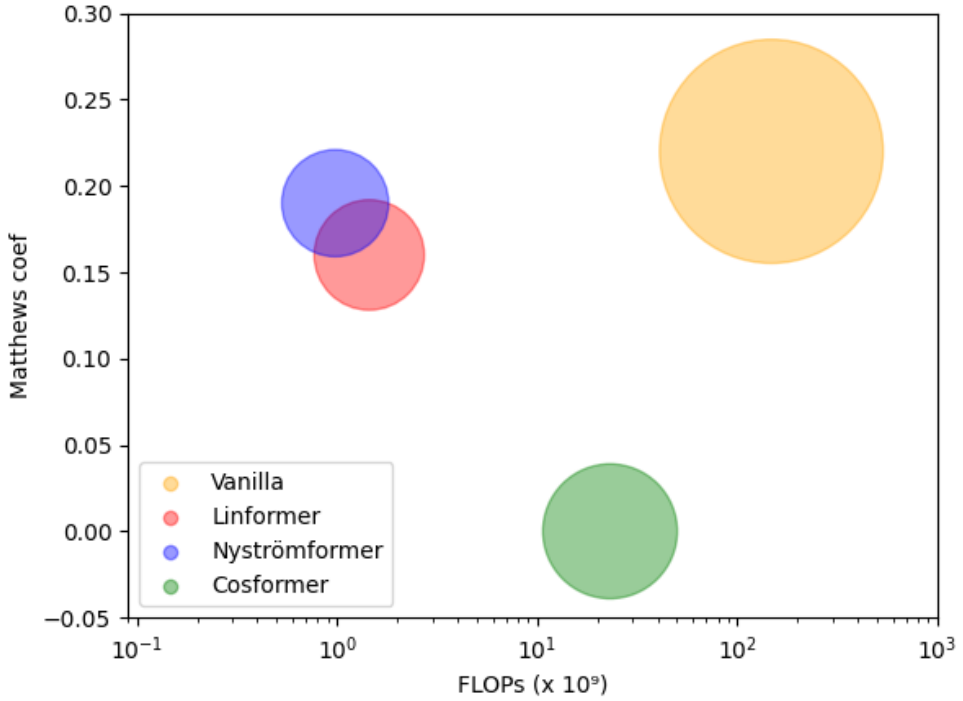


Figure 5.3: Performance vs FLOPs plotting. The FLOPs axis is in log scale. The size of the models circles is proportional to their required memory.

it indicates how Cosformer indeed fails to learn, achieving worse metrics than the other models. In conclusion, the results indicate that, empirically, the most effective model was Nyströmformer.

Now we combine the empirical results with the theoretical basis of our models to derive further conclusions. Before going into detail, we must emphasize two facts. First, Nyströmformer actually converged, since loss increased during the validations, and further training would not lead to better results. Regarding Linformer, its horizontal behaviour does not confirm that neither the model has fully converged or that it may become better with more epochs. Both possibilities are present and a larger training should be realized to determine whether it would surpass Nytsrömformer.

Secondly, our model is quite limited due to the number of attentions blocks and heads, and the training configuration may not be the ideal. The same reasoning applied to Cosformer regarding the configuration set up can be applied to Nyströmformer and Linformer, meaning that its configuration can impact their results. In any case, from now on we ignore the possibly derived consequences of

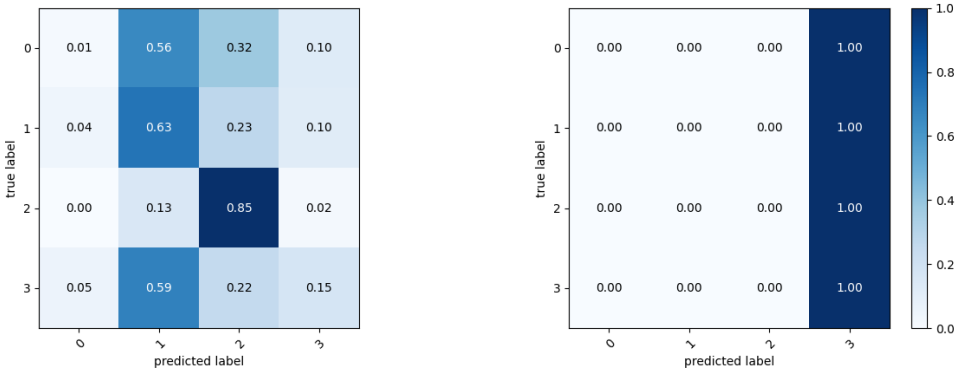
a non optimal configuration and work with the obtained results.

We hypothesize some fundamental reasons behind the Nyströmformer and Linformer reformulations to justify the results obtained. On the one hand, Linformer uses two extra projection matrices to reduce the dimension of the K and V matrices. This implies an extra layer, which increases the number of learnable parameters and may increase the capacity of the model to extract some specific features. Actually, it has been found that the usage of convolutionals layers prior to the self attention block on vision transformers can boost their performance [36]. Even though we are only considering linear projection, these extra parameters could boost the learning capacity of our model with a role similar to the convolutional layers, also delaying the convergence of the model. Exploiting this characteristic, it may be possible to further fine-tune the model, eventually surprassing Nyströmformer.

On the other hand, for Nyströmformer we suggest that the results obtained are a consequence of the landmark selection. This procedure may act as a sparse attention mechanism, reducing the number of attention scores computed and thus focusing on specific parts of the video. Since the video frames taken into account have no specific motive, we may be losing important information and the model capacity to learn may be affected. But the risk of hindering the capacity of the model implies a faster convergence. Therefore, in our experiment, Nytsrömformer was the only model that could be considered as fully trained.

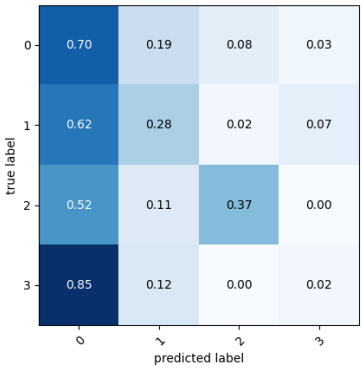
With all this information, we can answer the considered question *Which self attention mechanism has the best performance/computational cost trade off for video classification?* The results indicate that Nyströmformer is the best candidate. The hightest Matthews coefficient is obtained from it and its memory requirements are the lowest. Nevertheless, we should consider the potential of Linformer. The theory suggests that Linformer may be a suitable mechanism for computer vision tasks, and larger experiments to test our hypothesis might be realized. Contrarily, we would directly discard Cosformer, since the argued locality enforcement seems to prevent the model from learning temporal dependencies. Moreover, it also presents higher memory requirements, being the less attractive choice.

We suggest the reproduction of this experiment with an appropriate Transformer configuration and training hyperparameter search together with a more powerful hardware to confirm our results. Also, an interesting path of research would be to test the usage of Linformer to less expensive computer vision tasks, such as Image classification. In this way, our hypothesis of the additional projection matrices functioning in an analogous way as suggested in [36] would be tested.

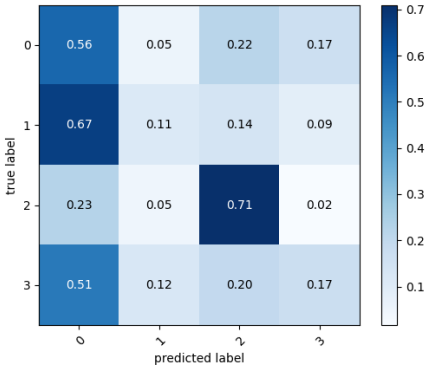


(a) Vanilla

(b) Cosformer



(c) Linformer



(d) Nytsrömformer

Figure 5.4: Confusion matrices for the different attention mechanisms. We observe how Cosformer fails to learn, predicting always the minority class. Nystömformer and Linformer obtain similar results, with a tendency to overfit the main class (i.e. the 0 class).

Chapter 6

Conclusions

With this work, we have satisfied both the theoretical and experimental contemplated objectives at Sect. 1.1. To begin with, we constructed a theoretical basis based on kernels and presented some of the properties crucial for the construction of self attention. The construction itself was carefully performed from scratch, giving great detail in both the conceptual concepts behind self attention and its mathematical formulation. The latter allowed us to realize a computational analysis of self attention and shed light to its main drawback, its quadratic dependence on the input length. Different approaches to overcome it were presented, discussing in great detail their respective reformulations. With all the aforementioned work, we fulfilled our theoretical objective of constructing self attention, learning its fundamentals and beyond along the process.

For the experimental part, we realized a large and complex work to develop our experiment. We made an important investment of time to learn the specific libraries and techniques necessary to carry out a model development of this type. The code demanded us a high implication and we realized several tryouts until the experimental setup was fully built up. Nevertheless, at the end, we were able to realize a simple yet insightful comparison among the efficient attention mechanisms presented at Chpt. 3. Nyströmformer obtained the best results, closely followed by the Linformer, and the theoretical reasons behind these behaviours were discussed. In any case, we are conscious of the limitations of our work and that the conclusions derived from it would need further confirmation. Our goal was ambitious and our resources modest, and more complete experimentation would be needed to extract decisive conclusions. Despite this, we hope our work may be a good baseline and could provide some hints for future research.

Bibliography

[1] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[2] Zhen Qin et al. *cosFormer: Rethinking Softmax in Attention*. 2022. arXiv: 2202.08791 [cs.CL].

[3] Yunyang Xiong et al. *Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention*. 2021. arXiv: 2102.03902 [cs.CL].

[4] Sinong Wang et al. *Linformer: Self-Attention with Linear Complexity*. 2020. arXiv: 2006.04768 [cs.LG].

[5] Dima Damen et al. *Scaling Egocentric Vision: The EPIC-KITCHENS Dataset*. 2018.

[6] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[7] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG].

[8] Thomas Wood. *A complte guide to Natural Language Processing*. URL: https://www.deeplearning.ai/resources/natural-language-processing/.

[9] Shervin Minaee et al. *Deep Learning Based Text Classification: A Comprehensive Review*. 2021. arXiv: 2004.03705 [cs.CL].

[10] Wenpeng Yin et al. *Comparative Study of CNN and RNN for Natural Language Processing*. 2017. arXiv: 1702.01923 [cs.CL].

[11] Zihao Wang and Lei Wu. *Theoretical Analysis of Inductive Biases in Deep Convolutional Networks*. 2023. arXiv: 2305.08404 [cs.LG].

[12] Anurag Arnab et al. *ViViT: A Video Vision Transformer*. 2021. arXiv: 2103.15691 [cs.CV].

[13] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[14] Yao-Hung Hubert Tsai et al. *Transformer Dissection: A Unified Understanding of Transformer's Attention via the Lens of Kernel*. 2019. arXiv: 1908.11775 [cs.LG].

[15] Joe Suzuki. *Kernel Methods for Machine Learning with Math and Python*. Springer Singapore, 2022, pp. 1–7. ISBN: 978-981-19-0401-1. DOI: https://doi.org/10.1007/978-981-19-0401-1.

[16] Biao Zhang, Ivan Titov, and Rico Sennrich. *Sparse Attention with Linear Units*. English. 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021 ; Conference date: 07-11-2021 Through 11-11-2021. Nov. 2021. URL: https://2021.emnlp.org/.

[17] Thomas Wood. *What is the Softmax Function?* URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[18] Michalis K. Titsias. *One-vs-Each Approximation to Softmax for Scalable Estimation of Probabilities*. 2016. arXiv: 1609.07410 [stat.ML].

[19] Yi Tay et al. *Efficient Transformers: A Survey*. 2022. arXiv: 2009.06732 [cs.LG].

[20] H. S. M. (Harold Scott Macdonald) Coxeter. *Geometry revisited*. eng. 6th print. New mathematical library ; 19. Mathematical Association of America, 1988, pp. 42–43. ISBN: 0883856190.

[21] R.I. Arriaga and S. Vempala. *An algorithmic theory of learning: robust concepts and random projection*. 1999. DOI: 10.1109/SFFCS.1999.814637.

[22] William Johnson and J. Lindenstrauss. *Extensions of Lipschitz mappings into a Hilbert space*. Jan. 1982.

[23] Petros Drineas and Michael W. Mahoney. *On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*. 2005.

[24] Shusen Wang. *On The Lower Bounds of The Nyström Method*. 2013. arXiv: 1303.4207. URL: http://arxiv.org/abs/1303.4207.

[25] Mostafa Kafaei Razavi et al. *A New Iterative Method for Finding Approximate Inverses of Complex Matrices*. 2014.

[26] Dinghan Shen et al. *Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms*. 2018. arXiv: 1805.09843 [cs.CL].

[27] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[28] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG].

[29] Chulhee Yun et al. *Are Transformers universal approximators of sequence-to-sequence functions?* 2020. arXiv: 1912.10077 [cs.LG].

[30] Zhi-Hua Zhou. *Machine Learning*. Springer Singapore, 2021, pp. 108–113. ISBN: 978-981-15-1967-3.

[31] Anqi Mao, Mehryar Mohri, and Yutao Zhong. *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. 2023. arXiv: 2304.07288 [cs.LG].

[32] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[33] Shiliang Sun et al. *A Survey of Optimization Methods from a Machine Learning Perspective*. 2019. arXiv: 1906.06821 [cs.LG].

[34] David Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation". In: *Mach. Learn. Technol.* 2 (Jan. 2008).

[35] Pierre Baldi et al. "Assessing the accuracy of prediction algorithms for classification: an overview ". In: *Bioinformatics* 16.5 (May 2000), pp. 412–424. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/16.5.412. eprint: https://academic.oup.com/bioinformatics/article-pdf/16/5/412/48836094/bioinformatics_16_5_412.pdf. URL: https://doi.org/10.1093/bioinformatics/16.5.412.

[36] Tete Xiao et al. *Early Convolutions Help Transformers See Better*. 2021. arXiv: 2106.14881 [cs.CV].