An Optimal Transport Kernel for Feature Aggregation and its Relationship to Attention

Grégoire Mialon, Dexiong Chen, Alexandre d'Aspremont, Julien Mairal

An Optimal Transport Kernel for Feature Aggregation and its Relationship to Attention

Grégoire Mialon*
Inria†‡
gregoire.mialon@inria.fr

Dexiong Chen*
Inria†
dexiong.chen@inria.fr

Alexandre d'Aspremont
CNRS - ENS‡
aspremon@ens.fr

Julien Mairal
Inria†
julien.mairal@inria.fr

June 29, 2020

Abstract

We introduce a kernel for sets of features based on an optimal transport distance, along with an explicit embedding function. Our approach addresses the problem of feature aggregation, or pooling, for sets that exhibit long-range dependencies between their members. More precisely, our embedding aggregates the features of a given set according to the transport plan between the set and a reference shared across the data set. Unlike traditional hand-crafted kernels, our embedding can be optimized for a specific task or data set. It also has a natural connection to attention mechanisms in neural networks, which are commonly used to deal with sets, yet requires less data. Our embedding is particularly suited for biological sequence classification tasks and shows promising results for natural language sequences. We provide an implementation of our embedding that can be used alone or as a module in larger learning models. Our code is freely available at `https://github.com/claying/OTK`.

1 Introduction

Many popular applications of machine learning such as natural language processing (NLP), computer vision, or bioinformatics rely on sets of features with positional information (sentences, pixels of an image, nodes of a graph, or biological sequences). These objects are delicate to manipulate due to potentially long-range and complex structural dependencies, or varying lengths. Before the resurgence of deep learning models, kernel methods have been widely used to handle such data [35]. In particular, a family of kernels used in biology and computer vision relies on the comparison of histograms of features [3, 21]. Yet, comparing histograms bin per bin can be restrictive, which has motivated more flexible kernel methods, through, *e.g.*, the creation of multi-resolution histograms [14], or by computing correspondences with more complex metrics. In this last context, the earth-mover's distance (which is another name for the 1-Wasserstein distance from optimal transport theory [29]) between histograms was investigated relatively early for its flexibility and robustness [34]. The main advantage of these metrics is that they do not require explicit computation of histograms but only comparisons between each pair of features. However, when large amounts of data are available, methods that optimize the representation to the task at hand are now preferred.

Among them, the concept of attention [2] was proposed to cope with long sentences in the context of neural machine translation. This mechanism allows the model to automatically search for parts of a source sentence

*Equal contribution.
†Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.
‡D.I., UMR 8548, École Normale Supérieure, Paris, France.

that are relevant for predicting the next word. A striking development of attention was the transformer [40], a neural network architecture relying mostly on attention mechanisms, which led to major progress in many NLP tasks [42] and to some extent in other fields relying on structured data such as computer vision [32] or bioinformatics [33]. A major drawback of these models however is their possibly prohibitive number of parameters: NLP state-of-the-art models such as T5 [30], may have up to 11 billion parameters. Moreover, training these models requires setting up an auxiliary learning task such as predicting the next word in a sentence, which is not always possible. Finally, some peculiarities of the transformer architecture, such as the learned dot-product self-attention or the role of the attention heads are now being questioned and investigated [31, 41, 45].

In this paper, we introduce a new kernel for sets of features, along with an embedding, based on mechanisms posterior to traditional kernel approaches. First, the embedding can be optimized for a given task in the fashion of [24], thus gaining the ability to become data- or task-adaptive. Second, the kernel embedding performs feature alignment and crucially relies on a by-product of optimal transport (OT), the transport plan. OT recently gained interest in machine learning and enjoys efficient solvers [10], which are compatible with GPU computation and back-propagation. OT has been used in the context of computer vision (as mentioned above), NLP [20] and graphs [37]. More recently, using the transport plan as an attention score was proposed for network embeddings to align some data modalities [8]. Our paper goes beyond this idea and uses transport plans as a principle for feature aggregation, or pooling. We demonstrate the effectiveness of our kernel with images, sentences and biological sequences and clarify its relationship to attention and the transformer architecture. Finally, we provide in the supplementary material an implementation of our embedding that can be used alone or as a module in larger learning models.

Summary of contributions. First, we propose a new kernel for sets of features, which provides a rich data representation based on optimal transport, along with an explicit embedding function. Second, we demonstrate the scalability and effectiveness of our approach on images, biological and natural language sequences in unsupervised and supervised settings. Finally, we provide an implementation of our embedding that can be used alone or as a module in larger learning models. Our code is freely available at `https://github.com/claying/OTK`.

2 Preliminaries

In this section, we revisit classical kernels for sets and some important results in optimal transport, which will be useful for the construction of our kernel.

2.1 Kernel Methods and Match Kernels

Kernel methods map data living in a space $\mathcal{X}$ to a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, associated to a positive definite (p.d.) kernel K through a mapping function $\varphi : \mathcal{X} \to \mathcal{H}$, such that $K(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{H}}$. In this paper, we handle sets of features living in $\mathbb{R}^d$ and we define

$$\mathcal{X} = \left\{ \mathbf{x} | \mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \text{ such that } \mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d \text{ for some } n \geq 1 \right\}.$$

Members of $\mathcal{X}$ are typically vectorial representations of local data structures, such as patches for natural images, or sentences for text. The length of $\mathbf{x}$ denoted by n may vary, which is not a problem since the methods we introduce may take a sequence of any size as input, while providing a fixed-size embedding. Popular tools for comparing sets of features are match kernels [23, 38], typically

$$K_{\text{match}}(\mathbf{x}, \mathbf{x}') := \frac{1}{n}\frac{1}{n'} \sum_{i=1}^{n} \sum_{j=1}^{n'} \kappa(\mathbf{x}_i, \mathbf{x}'_j) = \left\langle \frac{1}{n} \sum_{i=1}^{n} \varphi(\mathbf{x}_i), \frac{1}{n'} \sum_{j=1}^{n'} \varphi(\mathbf{x}'_j) \right\rangle_{\mathcal{H}},$$

where κ is a p.d. kernel and φ denotes its associated mapping to an RKHS $\mathcal{H}$. A match kernel simply compares all possible pairs of features of $\mathbf{x}$ and $\mathbf{x}'$. The right-hand term exhibits the feature map of K_{match},

which corresponds to a mean pooling in the RKHS. In this process, important information may be averaged (*e.g.*, in biology, rare and relevant patterns may be drowned in useless ones), or artificially strong matches can be made [16]. These issues can be addressed by weighting each comparison $\kappa(\mathbf{x}_i, \mathbf{x}'_j)$. The weights are typically independent from the data and may include domain knowledge [24]. In contrast, we will introduce adaptive weights reflecting whether a pair $(\mathbf{x}_i, \mathbf{x}'_j)$ is aligned before comparison or, put differently, whether comparing $\mathbf{x}_i$ and $\mathbf{x}'_j$ is relevant for the task.

2.2 Optimal Transport

More precisely, our weights will be based on the transport plan between $\mathbf{x}$ and $\mathbf{x}'$ seen as weighted point clouds or discrete measures, which is a by-product of the optimal transport problem. OT has indeed been widely used in alignment problems. Throughout the paper, we will refer to the Kantorovich relaxation of OT with entropic regularization, detailed for example in [29]. Let $\mathbf{a}$ in Δ^n (probability simplex) and $\mathbf{b}$ in $\Delta^{n'}$ be the weights of the discrete measures $\sum_i \mathbf{a}_i \delta_{\mathbf{x}_i}$ and $\sum_j \mathbf{b}_j \delta_{\mathbf{x}'_j}$ with respective locations $\mathbf{x}$ and $\mathbf{x}'$, where $\delta_{\mathbf{x}}$ is the Dirac at position $\mathbf{x}$. Let $\mathbf{C}$ in $\mathbb{R}^{n \times n'}$ be a matrix representing the pairwise costs for aligning the elements of $\mathbf{x}$ and $\mathbf{x}'$. The entropic regularized Kantorovich relaxation of OT from $\mathbf{x}$ to $\mathbf{x}'$ is

$$\min_{\mathbf{P} \in U(\mathbf{a}, \mathbf{b})} \sum_{ij} \mathbf{C}_{ij} \mathbf{P}_{ij} - \varepsilon H(\mathbf{P}), \tag{1}$$

where $H(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij} \log(\mathbf{P}_{ij} - 1)$ is the entropic regularization with parameter ε, which controls the sparsity of $\mathbf{P}$, and U is the space of admissible couplings between $\mathbf{a}$ and $\mathbf{b}$:

$$U(\mathbf{a}, \mathbf{b}) = \{\mathbf{P} \in \mathbb{R}_+^{n \times n'} : \mathbf{P}\mathbf{1}_n = \mathbf{a} \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} = \mathbf{b}\}.$$

In practice, $\mathbf{a}$ and $\mathbf{b}$ are uniform measures since we consider the mass to be evenly distributed between the points. $\mathbf{P}$ is called the transport plan, which carries the information on how to distribute the mass of $\mathbf{x}$ in $\mathbf{x}'$ with minimal cost. The objective is ε-strongly convex, such that (1) has a unique solution. It is typically solved using a matrix scaling procedure known as Sinkhorn's algorithm (see, *e.g*, [29]).

3 An Optimal Transport Based Kernel and its Embedding

In this section, we present an optimal transport based kernel and introduce an alternative that is more scalable and can be optimized for a given task.

3.1 An Attractive yet non Positive Definite Kernel

We are now ready to introduce a first version of the match kernel, whose objective is to adaptively weight the comparisons of features through a form of alignment given by the optimal transport plan.

Definition 3.1 (Optimal transport match kernel)**.** *Let $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$ be two sets of respective length n and n'. The Optimal Transport Match Kernel is defined as*

$$K_{OT}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}'), \kappa(\mathbf{x}, \mathbf{x}') \rangle := \sum_{i,j} \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')_{ij} \kappa(\mathbf{x}_i, \mathbf{x}'_j), \tag{2}$$

where $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}') \in U\left(1/n, 1/n'\right)$ is the solution to the regularized optimal transport problem (1) *between $\mathbf{x}$ and $\mathbf{x}'$, whose cost $\mathbf{C} \in \mathbb{R}^{n \times n\prime}$ has entries $\mathbf{C}_{ij} = -\kappa(\mathbf{x}_i, \mathbf{x}'_j)$.*

When $\varepsilon = 0$, K_{OT} is equivalent to the 2-Wasserstein distance associated to the distance d_κ induced by κ—defined as $d_\kappa^2(u, v) = \kappa(u, u) - 2\kappa(u, v) + \kappa(v, v)$ for any u, v—in the following sense:

$$W_2^2(\mathbf{x}, \mathbf{x}') = \min_{\mathbf{P}_\kappa \in U\left(\frac{1}{n}, \frac{1}{n\prime}\right)} \langle \mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \rangle = \frac{1}{n} \sum_{i=1}^{n} \kappa(\mathbf{x}_i, \mathbf{x}_i) + \frac{1}{n'} \sum_{j=1}^{n'} \kappa(\mathbf{x}'_j, \mathbf{x}'_j) - 2K_{\mathrm{OT}}(\mathbf{x}, \mathbf{x}'). \tag{3}$$

K_{OT} performs well in practice when ε is not too small, as shown in Section 5, but suffers from three issues: (i) for small values of ε, it is not positive-definite (experiments exhibit negative eigenvalues in the Gram matrix); (ii) computing K_{OT} requires solving the transport problems between all pairs $(\mathbf{x}, \mathbf{x}')$ in the data set, which grows quadratically with the number of samples; (iii) K_{OT} cannot be optimized to the task at hand. We therefore introduce a positive definite kernel addressing these issues in the next subsection.

3.2 Building a Positive Definite Optimal-Transport-Based Kernel

The issue of positive definiteness of the 2-Wasserstein distance, corresponding to K_{OT} with $\varepsilon = 0$, is well known and has been studied (see [29] Section 8.3, [48] and Appendix A). We go further and address the three issues above at the same time. We propose a surrogate of K_{OT} inspired by the following observation:

$$\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}') := p \times \mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})\mathbf{P}_{\kappa}(\mathbf{x}', \mathbf{z})^{\top},$$

with $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{z}$ sets of features and $p = |\mathbf{z}|$, is a valid transport plan between $\mathbf{x}'$ and $\mathbf{x}$ thanks to the gluing lemma (see, *e.g*, [29]), and empirically, is a rough approximation of $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{x}')$. Other works explored the idea of computing the transport with respect to a common reference [28, 43] yet for the unregularized transport and with minimal use of the resulting embedding. By replacing the optimal transport plan $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{x}')$ in K_{OT} with $\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}')$, we get a new kernel that enjoys better properties than K_{OT}.

Definition 3.2 (Optimal Transport Kernel (OTK) and Embedding)**.** *The OTK is defined as*

$$K_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') := \langle \mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}'), \kappa(\mathbf{x}, \mathbf{x}') \rangle, \tag{4}$$

and its associated embedding $\Phi_{\mathbf{z}}$ *of* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *such that* $K_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') = \langle \Phi_{\mathbf{z}}(\mathbf{x}), \Phi_{\mathbf{z}}(\mathbf{x}') \rangle$ *is*

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times \left(\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})_1^{\top} \varphi(\mathbf{x}), \ldots, \mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})_p^{\top} \varphi(\mathbf{x}) \right) = \sqrt{p} \times \mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})^{\top} \varphi(\mathbf{x}),$$

where $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})_i$ *denotes the i-th column of* $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})$*, i.e the couplings between the elements of* $\mathbf{x}$ *and* $\mathbf{z_i}$*, and* $\varphi(\mathbf{x}) := [\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)]^{\top}$*, with* $\varphi : \mathbb{R}^d \to \mathcal{H}$ *the kernel embedding associated to* κ *and its RKHS* $\mathcal{H}$*. We design an element* $\mathbf{z}_i$ *of* $\mathbf{z}$ *as a "support" and* p *as number of supports.*

The OTK $K_{\mathbf{z}}$ solves the issues raised above: (i) it is p.d. since it can be cast as $\langle \Phi_{\mathbf{z}}(\mathbf{x}), \Phi_{\mathbf{z}}(\mathbf{x}') \rangle$; (ii) computing its Gram matrix requires computing only as many transport plans as samples (all the transports from the samples $\mathbf{x}$ to the reference $\mathbf{z}$); (iii) as we will show later, the parameter $\mathbf{z}$ can be adapted to a specific task. Regarding the interpretation of the embedding $\Phi_{\mathbf{z}}(\mathbf{x})$, the notion of pooling in the RKHS $\mathcal{H}$ of κ arises naturally if $p \leq n$. $\Phi_{\mathbf{z}}$ simultaneously embeds $\mathbf{x}$ to $\mathcal{H}^n$ (via φ), aligns (via $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})$), which is a mechanism akin to that of an attention layer, see Section 4, but also pools to $\mathcal{H}^p$. In other words, the elements of $\mathbf{x}$ are non-linearly embedded and then aggregated in "buckets", one for each element in the reference $\mathbf{z}$, given the values of $\mathbf{P}_{\kappa}(\mathbf{x}, \mathbf{z})$. This process is illustrated in Figure 1.

From infinite-dimensional kernel embedding to finite dimension. In some cases, $\varphi(\mathbf{x})$ is already finite-dimensional, which allows to compute the embedding $\Phi_{\mathbf{z}}(\mathbf{x})$ explicitly. This is particularly useful when dealing with large-scale data, as it allows us to use our method for supervised learning tasks without computing the Gram matrix, which grows quadratically with the number of samples. When φ is infinite- or high-dimensional, it is nevertheless possible to use an approximation based on the Nyström method [17], which provides an embedding $\psi : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}'_j) \rangle_{\mathbb{R}^k} \approx \kappa(\mathbf{x}_i, \mathbf{x}'_j).$$

Concretely, the Nyström method consists in projecting points from the RKHS $\mathcal{H}$ onto a linear subspace $\mathcal{F}$, which is parametrized by k anchor points $\mathcal{F} = \mathrm{Span}(\varphi(\mathbf{w}_1), \ldots, \varphi(\mathbf{w}_k))$. The corresponding embedding admits an explicit form $\psi(\mathbf{x}_i) = \kappa(\mathbf{w}, \mathbf{w})^{-1/2} \kappa(\mathbf{w}, \mathbf{x}_i)$, where $\kappa(\mathbf{w}, \mathbf{w})$ is the $k \times k$ Gram matrix of κ computed on the set $\mathbf{w} = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ of anchor points and $\kappa(\mathbf{w}, \mathbf{x}_i)$ is in $\mathbb{R}^k$. Then, there are several ways of learning

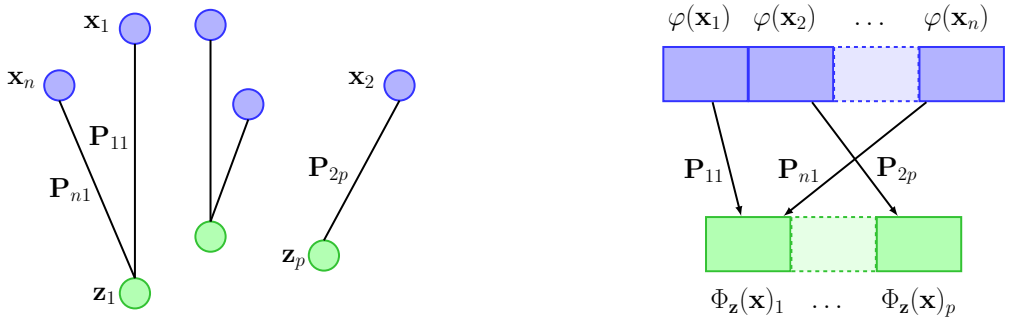

Figure 1: The input point cloud $\mathbf{x}$ is transported onto the reference $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_p)$ (left), yielding the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ used to aggregate the embedded features and form $\Phi_{\mathbf{z}}(\mathbf{x})$ (right).

the anchor points: (a) they can be chosen as random points from data; (b) they can be defined as centroids obtained by K-means, (c) they can be learned by back-propagation for a supervised task, see [24]. Once φ is replaced with ψ, the transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ with $\mathbf{z} \in \mathbb{R}^d$ in (4) can be reparametrized as $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z}')$, *i.e.* a transport whose cost is the opposite of the linear kernel and $\mathbf{z}' = \psi(\mathbf{z})$ in $\mathbb{R}^k$. By abuse of notation, we still use $\mathbf{z}$ for the new parametrization. The OTK embedding becomes simply

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times \mathbf{P}(\psi(\mathbf{x}), \mathbf{z})^\top \psi(\mathbf{x}) \in \mathbb{R}^{k \times p}, \tag{5}$$

with k the dimension of the Nyström embedding and p the number of support in $\mathbf{z}$. In our experiments, we will often use a Gaussian kernel so that (5) is the embedding used in practice. Next, we discuss how to learn the reference set $\mathbf{z}$.

3.3 Unsupervised and Supervised Learning of z

Unsupervised learning. Without labels, and in the fashion of the Nyström approximation, the p elements of $\mathbf{z}$ can be defined as the centroids obtained by K-means applied to features from available training sets in $\mathcal{X}$. The next lemma, proved in Appendix C, suggests another algorithm

Lemma 3.3 (Relation between $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{x}')$ and $\mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ when $\varepsilon = 0$)**.** *For any $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{z}$ in $\mathcal{X}$ with lengths n, n' and p,*

$$|W_2(\mathbf{x}, \mathbf{x}') - \underbrace{\langle \mathbf{P}_{\kappa,\mathbf{z}}(\mathbf{x}, \mathbf{x}'), d^2_\kappa(\mathbf{x}, \mathbf{x}') \rangle^{1/2}}_{W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')}| \leq 2 \min(W_2(\mathbf{x}, \mathbf{z}), W_2(\mathbf{x}', \mathbf{z})). \tag{6}$$

A corollary is a bound on the error term between W_2 and $W_2^{\mathbf{z}}$ for m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{m} \sum_{i=1}^{m} W_2^2(\mathbf{x}^i, \mathbf{z}). \tag{7}$$

Equation (6) shows that the distance $W_2^{\mathbf{z}}$ resulting from $K_{\mathbf{z}}$ is related to the Wasserstein distance W_2; yet, this relation should not be interpreted as an approximation error, as our goal is not to approximate W_2, but rather to develop a different p.d. kernel with good computational properties. The right-hand term in Equation (7) corresponds to the objective to minimize in the Wasserstein Barycenter [11] problem, which yields the mean of a set of empirical measures (here the $\mathbf{x}$'s) under the OT metric. The Wasserstein barycenter is therefore an attractive candidate for choosing $\mathbf{z}$. Both methods yield similar results as will be shown in Section 5 and Appendix D. Wasserstein barycenters are less theoretically grounded for non-linear kernels which further justifies our parametrization (5). The anchor points $\mathbf{w}$ and the references $\mathbf{z}$ may be computed

using similar algorithms; however, their mathematical interpretation differs as exposed above. The task of representing features (learning $\mathbf{w}$ in $\mathbb{R}^d$ for a specific κ) is decoupled from the task of aggregating (learning the reference $\mathbf{z}$ in $\mathbb{R}^k$), which is similar to the multilayer structure of neural networks.

Supervised learning. As mentioned in Section 2, $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$ is computed using Sinkhorn's algorithm, recalled in Appendix A, which can be easily adapted to batches of samples $\mathbf{x}$, with possibly varying lengths, leading to GPU-friendly forward computations of the OTK embedding $\Phi_{\mathbf{z}}$. More important, all Sinkhorn's operations are differentiable, which enables $\mathbf{z}$ to be optimized through back-propagation, thus paving the way for supervised learning in the context of empirical risk minimization. Moreover, a small number of Sinkhorn iterations is sufficient in practice to compute $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$. Since the anchors $\mathbf{w}$ in the embedding layer below can also be learned end-to-end [24], the OTK is a module that can be injected into any deep network, as demonstrated in our experiments.

3.4 Extensions

Integrating positional information into the OTK. The discussed kernels do not take the position of the features into account, which may be problematic when dealing with structured data such as images or sentences. To this end, we borrow the idea of convolutional kernel networks (CKN) [24, 26], *i.e.* to penalize the similarity exponentially with the positional distance between a pair of elements in the sequences. More precisely, we multiply κ by this positional term:

$$\kappa'(\mathbf{x}_i, \mathbf{x}'_j) = \kappa(\mathbf{x}_i, \mathbf{x}'_j) \times e^{-\frac{1}{\sigma^2_{\text{pos}}}(i/n - j/n')^2}.$$

and replace it in the OTK. With similarity weights based *both* on content and position, the OTK can be viewed as a generalization of the CKNs (whose similarity weights are based on position only), with feature alignment based on optimal transport. The details on the resulting embedding can be found in Appendix B. When dealing with multi-dimensional objects such as images, we just replace the index scalar i with an index vector of the same spatial dimension as the object, representing the positions of each dimension.

Using multiple references. A naive reconstruction using different references $\mathbf{z}^1, \dots, \mathbf{z}^q$ in $\mathcal{X}$ may yield a better approximation of the transport plan. In this case, the embedding of $\mathbf{x}$ becomes

$$\Phi_{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}) = {}^1\!/\!_{\sqrt{q}}\left(\Phi_{\mathbf{z}^1}(\mathbf{x}), \dots, \Phi_{\mathbf{z}^q}(\mathbf{x})\right), \tag{8}$$

with q the number of references (the factor ${}^1\!/\!_{\sqrt{q}}$ comes from the mean). The references do not necessarily have the same number of supports $\mathbf{z_i}$. Using relation (6) (see Appendix C for details), we can obtain an error bound similar to (7) for a data set of m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ and q references. To choose multiple references, we tried a K-means algorithm with 2-Wasserstein distance for assigning clusters, and we updated the centroids as in the single-reference case. We observe in Section 5 that using multiple references is particularly useful when optimizing $\mathbf{z}$ with supervision.

4 On the Relationship between the OTK and Self-Attention

Since our embedding and the transformer architecture, recalled in Appendix A, share the same type of inductive bias, *i.e*, aggregating features relying on similarity weights, we will now clarify their relationship. Our OTK embedding is arguably simpler as showed in Table 1, and may compete in some cases with the transformer self-attention as illustrated in Section 5.

Shared reference versus self-attention. The embedding Φ_A provided by a transformer layer with attention weights $\mathbf{W} \in \mathbb{R}^{n \times n}$ and values $\mathbf{V} \in \mathbb{R}^{n \times d}$, can be written (see Appendix A)

$$\Phi_A(\mathbf{x}) = \left(\mathbf{W}(\mathbf{x})_1^\top \mathbf{V}(\mathbf{x}), \dots, \mathbf{W}(\mathbf{x})_n^\top \mathbf{V}(\mathbf{x})\right) = \mathbf{W}(\mathbf{x})^\top \mathbf{V}(\mathbf{x}).$$

Table 1: Relationship between $\Phi_{\mathbf{z}}$ and transformer self-attention. k depends on how the transformer integrates positional information. q is the number of references / attention heads, d, the dimension of the embeddings, p, the number of supports in $\mathbf{z}$ and n the sequence length. Typically, $p \ll d$. In recent transformer architectures, positional encoding requires learning additional parameters ($\sim qd^2$).

	Transformer	$\Phi_{\mathbf{z}}$
Attention score	$\mathbf{W} = W^\top Q$	$\mathbf{P}$
Size of attention	$O(n^2)$	$O(np)$
Alignment w.r.t	$\mathbf{x}$ itself	$\mathbf{z}$
Learned + Shared	W and Q	$\mathbf{z}$
Nonlinear mapping	Feed-forward	φ or ψ
Position encoding	$k(t_i, {t'}_j)$	$e^{-\frac{1}{\sigma^2_{\text{pos}}}(\frac{i}{n} - \frac{j}{n'})^2}$
Nb. parameters	$\sim qd^2$	qpd
Supervision	Needed	Not needed

There is therefore a correspondence between $\mathbf{V}$, $\mathbf{W}$ in the transformer and φ, $\mathbf{P}$ in Definition 3.2, yet also noticeable differences. On the one hand, our embedding $\Phi_{\mathbf{z}}$ aligns a given sequence $\mathbf{x}$ with respect to a reference $\mathbf{z}$, learned with or without supervision, and shared across the data set. On the other hand, Φ_A performs self-alignment (or self-attention): for a given $\mathbf{x}_i$, features are aggregated depending on a similarity score between $\mathbf{x}_i$ and the elements of $\mathbf{x}$ only. The similarity score is a matrix product between queries Q and keys K matrices, learned with supervision and shared across the data set. In Section 5, we will show that supervised $\Phi_{\mathbf{z}}$ approaches a fully-trained transformer at a cheaper memory and parameter cost (see respectively size of attention and number of parameters in Table 1). In this regard, our work complements a recent line of research questioning the dot-product, learned self-attention [31, 45]. Note that self-attention-like weights can also be obtained with the OTK by computing $\mathbf{P}(\mathbf{x}, \mathbf{z}_i)\mathbf{P}(\mathbf{x}, \mathbf{z}_i)^\top$ for each reference i.

Position smoothing and relative positional encoding. For transformers, an absolute positional encoding was originally integrated to the word embeddings [40]; yet, relative positional encoding [12] is a current standard for integrating positional information: the position offset between the query element and a given key can be injected in the attention score [39], which is equivalent to our approach. The link between CKNs and OTK, provided by this positional encoding, stands in line with a recent line of research casting attention and convolution into a unified framework [1]. In particular, [9] shows that attention learns convolution in the setting of image classification: the kernel pattern is learned at the same time as the filters.

Multiple references and attention heads. In the transformer architecture, the succession of blocks composed of an attention layer followed by a fully-connected layer is called a head, with each head potentially focusing on different parts of the input. Successful architectures have a few heads in parallel. The outputs of the heads are then aggregated to output a final embedding. A layer of the OTK with non-linear kernel κ can be seen as a transformer block, with the references playing the role of heads. As some recent work questions the role of attention heads [41, 27], exploring the content of our learned references $\mathbf{z}$ may provide another perspective on this question.

5 Experiments

We demonstrate the effectiveness of the OTK embedding in biology, natural language processing and image classification tasks in unsupervised and supervised settings. Although K_{OT} in (2) and its surrogate $K_{\mathbf{z}}$ in (4) are of interest, their lack of scalability – they require to compute the Gram matrix, which is quadratic in the number of samples – makes them less suited to large data sets, unlike our explicit embedding $\Phi_{\mathbf{z}}$. Nevertheless, a brief study of their performance can be found in Appendix D.

Table 2: Classification accuracy (top 1/5/10) on test set for SCOP 1.75 for different combinations of (number of references q $\times$ number of supports p). The accuracies are averaged from 10 different runs. DeepSF is a CNN with 10 convolutional layers. The OTK outperforms all baselines.

Unsupervised					
Nb filters	CKN [6]			OTK (1×50)	OTK (1×100)
128	64.7/86.3/91.6			77.5/91.7/94.5	**79.4/92.4/94.9**
1024	82.2/92.8/95.2			84.6/95.0/**97.0**	**85.7/95.3**/96.7
Supervised					
Nb filters	DeepSF [15]	CKN [6]	RKN [7]	OTK (1×50)	OTK (5×10)
128	73.0/90.3/94.5	76.3/92.2/95.3	77.8/92.9/95.5	82.8/93.9/96.2	**84.7/94.7/96.5**
512		84.1/94.3/96.4	85.3/95.0/96.5	88.4/95.8/97.1	**88.7/95.9/97.3**

Table 3: Results for prediction of chromatin profiles on the DeepSEA dataset. The metrics are area under ROC (auROC) and area under PR curve (auPRC), averaged over 919 chromatin profiles. The accuracies are averaged from 10 different runs. Armed with the positional encoding (PE) described in Section 3, the OTK outperforms the state-of-the-art model and an OTK with the PE proposed in [40].

Method	DeepSEA [47] (3-layer-CNN)	OTK + Sinusoidal PE [40]	OTK + Our PE
auROC	0.933	0.917	**0.936**
auPRC	0.342	0.311	**0.360**

Influence of z on the OTK embedding. The OTK embedding $\Phi_{\mathbf{z}}$ defined in (8) is characterized by the number of references q and the number p of features (supports) in each reference set. As mentioned in Section 3, we investigate several algorithms for learning the references without supervision. The discussed results can be found in Appendix D. Using more references and increasing the number of supports generally yields better results at the expense of a larger computational cost and K-means turned out to be a simple and effective approach for learning the references.

Protein fold classification on SCOP 1.75. We follow the protocol described in [15] for this classical bioinformatics task. The training, validation and test set contain respectively 14699, 2013 and 2533 sequences with 1195 labels. The sequence lengths vary from tens to thousands and each element of a sequence is a vector of 45 dimensions. More details can be found in Appendix D.3. The sequences are encoded with a Gaussian kernel as in CKNs [6]. While a global average pooling operation is used to aggregate the kernel embeddings in CKNs, the OTK embedding (5) performs an adaptive pooling. Different numbers of anchor points (128, 512 and 1024) are considered in the Nyström approximation. In the unsupervised setting, we benchmark our features against the regular, unsupervised state-of-the-art CKN features [7]. As shown in Table 2, the OTK embedding clearly outperforms the CKN features. In the supervised setting, we compare our optimized features to three state-of-the-art features for this task, obtained by recurrent kernel networks (RKN) [7] and CKNs, both learned with supervision, and a more traditional model based on CNNs, DeepSF [15]. Our method outperforms all baselines as shown in Table 2. Note how our unsupervised features are as good if not better than the supervised baselines. Complementary results on the effect of $\mathbf{z}$, and comparison with Wasserstein barycenter for learning $\mathbf{z}$ and error bars can be found in Appendix D.3.

Detection of chromatin profiles. Predicting the chromatin features such as transcription factors (TF) binding from genomic sequences has been studied extensively in the recent years. CNNs with pooling operations have been shown effective for this task. We show that attention-based models like our OTK can achieve competitive results. The DeepSEA dataset [47] consists in simultaneously predicting 919 chromatin profiles. There are 4.4 million, 8000 and 455024 samples respectively in training, validation and test set. Each sample consists of a 1000-bp DNA sequence from the human GRCh37 reference. The DNA characters

Table 4: Unsupervised and supervised classification accuracies for SST-2 reported on standard validation set. The accuracies are averaged from 10 different runs. The features are 30 words whose embedding has dimension 768. The reference is computed using K-means, with (number of references q × number of supports p). Our unsupervised OTK improves the pre-trained features, and its supervised counterpart trained on the pre-trained features gets closer to the fine-tuned features.

	[CLS]	Flatten	Mean pooling	OTK (1×300)	OTK (1×30)
BERT Features	BERT [13] Baselines			Unsupervised	Supervised
Pre-trained	84.6	84.9	85.3	86.8	**87.7**
Fine-tuned	90.3	**91.0**	90.8	90.9	-

are represented using the one-hot encoding and each sequence is thus represented as a 1000×4 binary matrix. As this problem is very imbalanced for a given profile, learning an unsupervised model could require an extremely large number of parameters. We thus use our supervised OTK in (5) as an adaptive pooling layer and inject it into a deep neural network model, between a convolutional layer and a fully connected layer. We compare it to the state-of-the art model [47], a CNN with 3 convolutional layers, in Table 3. In contrast to a typical transformer which would have stored a 1000×1000 matrix, our attention score, with a reference of size 64, is only 1000×64. Realizing that position encoding is crucial for this task, we also compare our encoding to the sinusoidal encoding introduced in the transformer [40] and find that ours is more effective here. More details about model architectures and training can be found in Appendix D.4.

Sentiment analysis on Stanford Sentiment Treebank. SST-2 [36] belongs to the classical NLP GLUE benchmark [42] and consists in predicting whether a movie review is positive. The train, validation and test sets contain respectively 67349, 872 and 1821 reviews. More details can be found in Appendix D. The test predictions need to be submitted on the GLUE leaderboard, so that we report accuracies on the validation set used as a test set. State-of-the-art accuracies are usually obtained after supervised fine-tuning of pre-trained transformers on the data set. When the user does not have access to GPUs for fine-tuning, training a linear model on the pre-trained features still yields good results. The word features fed to $\Phi_{\mathbf{z}}$ (5) are provided by the HuggingFace implementation [46] of the famous transformer BERT [13]. In the unsupervised setting, we benchmark against various combinations of the BERT features. [CLS] denotes the BERT embedding used for classification. The adaptive pooling of $\Phi_{\mathbf{z}}$ notably improves the BERT pre-trained features, as shown in Table 4. Complementary results on the effect of $\mathbf{z}$ can be found in Appendix D. In the supervised setting, an OTK embedding with Gaussian kernel, trained on BERT pre-trained features, gets closer to the fully fine-tuned BERT.

Image classification on CIFAR-10. Here we use CIFAR-10 features, *i.e.* 60000 images with 32×32 pixels and 10 classes encoded using a two-layer CKN [24], one of the baseline architectures for unsupervised learning of CIFAR-10, and evaluate on the standard test set. The very best configuration of the CKN yields a small number (3×3) of high-dimensional (16384) patches and an accuracy of 85.8%. Because our embedding is designed for larger sets of features, it is more consistent to illustrate it on a configuration which performs slightly less but provides more patches (16×16). While the CKN uses a Gaussian pooling (with pooling size equal to 6) after a 2-layer convolutional kernel, our OTKs (5) performs an adaptive pooling. The results are shown on Table 5. Again, without supervision, the adaptive pooling of the CKN features by the OTK notably improves their performance. We notice that the position encoding is very important to this task, which substantially improves the performance of its counterpart without it. More details can be found in Appendix D.2.

Table 5: Unsupervised classification with CIFAR-10 for various features extracted from 2-layer unsupervised CKNs with different numbers of filters. The accuracies are averaged from 10 different runs. The unsupervised OTK has one reference with 64 supports learned with K-means, with or without our position encoding (PE). Our OTK embedding notably improves the base features.

Dataset	Flatten	Mean pooling	Gaussian pooling [24]	OTK	OTK (with PE)
$16 \times 16 \times 256$	73.1	64.9	77.5	77.9	**81.4**
$16 \times 16 \times 1024$	76.1	73.4	82.0	80.1	**83.2**

Acknowledgments

JM, GM and DC were supported by the ERC grant number 714381 (SOLARIS project) and by ANR 3IA MIAI@Grenoble Alpes, (ANR-19-P3IA-0003). AA would like to acknowledge support from the *ML and Optimisation* joint research initiative with the *fonds AXA pour la recherche* and Kamet Ventures, a Google focused award, as well as funding by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute). DC and GM thank Laurent Jacob, Louis Martin, François-Pierre Paty and Thomas Wolf for useful discussions.

References

[1] J.-M. Andreoli. Convolution, attention and structure embedding. In *arXiv preprint arXiv: 1905.01289v5*, 2019.

[2] D. Bahdanau, K. Cho, and J. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[3] A. Barla, F. Odone, and A. Verri. Histogram intersesction kernel for image classification. In *Proceedings 2003 International Conference on Image Processing*, pages III–513. IEEE, 2003.

[4] R. Bhatia, T. Jain, and Y. Lim. On the bures-wasserstein distance between positive definite matrices. *Expositiones Mathematicae*, 2018.

[5] A. Buades, B. Coll, and J.-M. Morel. Non-Local Means Denoising. *Image Processing On Line*, 1:208–212, 2011.

[6] D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, pages 35(18):3294–3302, 2019.

[7] D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[8] L. Chen, G. Wang, C. Tao, D. Shen, P. Cheng, X. Zhang, W. Wang, Y. Zhang, and L. Carin. Improving textual network embedding with global attention via optimal transport. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[9] J.-B. Cordonnier, A. Loukas, and M. Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations (ICLR)*, 2020.

[10] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[11] M. Cuturi and A. Doucet. Fast computation of wasserstein barycenters. In *International Conference on Machine Learning (ICML)*, 2013.

[12] Z. Dai, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[14] K. Grauman and T. Darell. The pyramid match kernel: Efficient learning with sets of features. *Journal of Machine Learning Research (JMLR)*, pages 725–760, 2007.

[15] J. Hou, B. Adhikari, and J. Cheng. Deepsf: deep convolutional neural network for mappingprotein sequences to folds. *Bioinformatics*, pages 34(8):1295–1303, 2019.

[16] H. Jégou, D. Matthijs, and C. Schmid. On the burstiness of visual elements. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[17] C. K.I. Williams and S. Matthias. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2001.

[18] N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] P. P. Kuksa, P.-h. Huang, and V. Pavlovic. Scalable algorithms for string kernels with inexact matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009.

[20] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning (ICML)*, 2015.

[21] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[22] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

[23] S. Lyu. Mercer kernels for object recognition with local features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[24] J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[25] J. Mairal. Cyanure: An open-source toolbox for empirical risk minimization for python, C++, and soon more. *arXiv preprint arXiv:1912.08165*, 2019.

[26] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[27] P. Michel, O. Levy, and G. Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[28] Q. Mérigot, A. Delalande, and F. Chazal. Quantitative stability of optimal transport maps and linearization of the 2-wasserstein space. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[29] G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–206, 2019.

[30] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. In *arXiv preprint arXiv 1910.10683*, 2019.

[31] A. Raganato, Y. Scherrer, and T. Jörg. Fixed encoder self-attention patterns in transformer-based machine translation. In *arXiv preprint arXiv: 2002.10260*, 2020.

[32] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levsakaya, and J. Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[33] A. Rives, S. Goyal, J. Meier, D. Guo, M. Ott, C. L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. In *bioRxiv 622803*, 2019.

[34] Y. Rubner, C. Tomasi, and L. J. Guibad. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[35] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[36] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[37] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[38] G. Tolias, Y. Avrithis, and H. Jégou. To aggregate or not to aggregate: Selective match kernels for image search. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2013.

[39] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

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

[41] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[42] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: a multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019.

[43] W. Wang, D. Slepcev, S. Basu, J. A. Ozolek, and G. K. Rohde. A linear optimal transportation framework for quantifying and visualizing variations in sets of images. *International Journal of Computer Vision*, 101(2):254–269, 2013.

[44] X. Wang, R. B. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[45] Y. Weiqiu, S. Sun, and M. Iyyer. Hard-coded gaussian attention for neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[46] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. In *arXiv preprint arXiv: 1910.03771*, 2019.

[47] J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

[48] J. Zhou and O. G. Troyanskaya. On the definiteness of earth mover's distance and its relation to set intersection. *IEEE Transactions on Cybernetics*, 48(11):3184–3196, 2018.

Appendix

Appendix A provides some background on notions used throughout the paper; Appendix B adds details on the implementation and foundation of our OTK; Appendix C contains the proofs skipped in the paper; Appendix D provides details on our experimental protocol for reproducibility and additional results.

A Background

This section provides some background on attention and transformers, Sinkhorn's algorithm and the relationship between optimal transport based kernels and positive definite histogram kernels.

A.1 Attention and transformers

We clarify the concept of attention — a mechanism yielding a context-dependent embedding for each element of $\mathbf{x}$ — as a special case of non-local operations [44, 5], so that it is easier to understand its relationship to the OTK. Let us assume we are given a set $\mathbf{x} \in \mathcal{X}$ of length n. A non-local operation on an element $\mathbf{x}_i$ of $\mathbf{x}_i$ is a function $\Phi : \mathcal{X} \mapsto \mathcal{X}$ such that

$$\Phi(\mathbf{x})_i = \sum_{j=1}^{n} w(\mathbf{x}_i, \mathbf{x}_j) v(\mathbf{x}_j) = \mathbf{W}(\mathbf{x})_i^\top \mathbf{V}(\mathbf{x}),$$

where $\mathbf{W}(\mathbf{x})_i$ denotes the i-th column of $\mathbf{W}(\mathbf{x})$, a weighting function, and $\mathbf{V}(\mathbf{x}) = [v(\mathbf{x}_1), \ldots, v(\mathbf{x}_n)]^\top$, an embedding. In contrast to operations on local neighborhood such as convolutions, non-local operations theoretically account for long range dependencies between elements in the set. In attention and the context of neural networks, w is a *learned* function reflecting the *relevance* of each other elements $\mathbf{x}_j$ with respect to the element $\mathbf{x}_i$ being embedded and given the task at hand. In the context of the paper, we compare to a type of attention coined as *dot-product self-attention*, which can typically be found in the encoder part of the transformer architecture [40]. Transformers are neural network models relying mostly on a succession of an attention layer followed by a fully-connected layer. Transformers can be used in sequence-to-sequence tasks — in this setting, they have an encoder with self-attention and a decoder part with a variant of self-attention —, or in sequence to label tasks, with only the encoder part. The paper deals with the latter. The name self-attention means that the attention is computed using a dot-product of linear transformations of $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{x}$ attends to itself only. In its matrix formulation, dot-product self-attention is a non-local operation whose matching vector is

$$\mathbf{W}(\mathbf{x})_i = \text{Softmax}\left(\frac{W_Q \mathbf{x}_i \mathbf{x}^\top W_K^\top}{\sqrt{d_k}}\right),$$

where $W_Q \in \mathbb{R}^{n \times d_k}$ and $W_K \in \mathbb{R}^{n \times d_k}$ are learned by the network. In order to know which $\mathbf{x}_j$ are relevant to $\mathbf{x}_i$, the network computes scores between a query for $\mathbf{x}_i$ ($W_Q \mathbf{x}_i$) and keys of all the elements of $\mathbf{x}$ ($W_K \mathbf{x}$). The softmax turns the scores into a weight vector in the simplex. Moreover, a linear mapping $\mathbf{V}(\mathbf{x}) = W_V \mathbf{x}$, the values, is also learned. W_Q and W_K are often shared [18]. A drawback of such attention is that for a sequence of length n, the model has to store an attention matrix $\mathbf{W}$ with size $O(n^2)$. More details can be found in [40].

A.2 Sinkhorn's Algorithm: Fast Computation of $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$

Without loss of generality, we consider here κ the linear kernel. We recall that $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ is the solution of an optimal transport problem, which can be efficiently solved by Sinkhorn's algorithm [29] involving matrix

multiplications only. Specifically, Sinkhorn's algorithm is an iterative matrix scaling method that takes the opposite of the pairwise similarity matrix $\mathbf{K}$ with entry $\mathbf{K}_{ij} := \langle \mathbf{x}_i, \mathbf{z}_j \rangle$ as input $\mathbf{C}$ and outputs the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z}) = \text{Sinkhorn}(\mathbf{K}, \varepsilon)$. Each iteration step ℓ performs the following updates

$$\mathbf{u}^{(\ell+1)} = \frac{1/n}{\mathbf{E}\mathbf{v}^{(\ell)}} \quad \text{and} \quad \mathbf{v}^{(\ell+1)} = \frac{1/p}{\mathbf{E}^\top \mathbf{u}^{(\ell)}}, \tag{9}$$

where $\mathbf{E} = e^{\mathbf{K}/\varepsilon}$. Then the matrix $\text{diag}(\mathbf{u}^{(\ell)})\mathbf{E}\text{diag}(\mathbf{v}^{(\ell)})$ converges to $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ when ℓ tends to ∞. However when ε becomes too small, some of the elements of a matrix product $\mathbf{E}\mathbf{v}$ or $\mathbf{E}^\top \mathbf{u}$ become null and result in a division by 0. To overcome this numerical stability issue, computing the multipliers $\mathbf{u}$ and $\mathbf{v}$ is preferred (see *e.g.* [29, Remark 4.23]). This algorithm can be easily adapted to a batch of data points $\mathbf{x}$, and with possibly varying lengths via a mask vector masking on the padding positions of each data point $\mathbf{x}$, leading to GPU-friendly computation. More importantly, all the operations above at each step are differentiable, which enables $\mathbf{z}$ to be optimized through back-propagation. Consequently, this module can be injected into any deep networks.

A.3 On the Relationship Between Optimal Transport Match Kernel and Histogram Kernels

When features are living in a discrete set $\mathcal{F}$, it is classical to represent a set of features $\mathbf{x}$ as a histogram $\mathbf{H}(\mathbf{x}) = (\mathbf{H}_\mathbf{u}(\mathbf{x}))_{\mathbf{u}\in\mathcal{F}}$, with $\mathbf{H}_\mathbf{u}(\mathbf{x})$ the number of occurrences of the pattern $\mathbf{u}$ in $\mathbf{x}$. The spectrum kernel [21] used in biology computes the dot product between the normalized histograms $\hat{\mathbf{H}}(\mathbf{x}) = \mathbf{H}(\mathbf{x})/|\mathbf{x}|$ and $\hat{\mathbf{H}}(\mathbf{x}')$. Interestingly, this kernel can be rewritten [19]

$$K_{\text{spectrum}}(\mathbf{x}, \mathbf{x}') := \langle \hat{\mathbf{H}}(\mathbf{x}), \hat{\mathbf{H}}(\mathbf{x}') \rangle = \frac{1}{n}\frac{1}{n'} \sum_{i=1}^{n} \sum_{j=1}^{n'} \delta(\mathbf{x}_i, \mathbf{x}'_j),$$

where δ denotes the Dirac kernel. As the spectrum kernel performs exact matches between elements, more flexible variants where then proposed, either by allowing mismatches for sequence data [22], or by using a Gaussian kernel for comparing features [6]. A related version of K_{spectrum} is the histogram intersection kernel [3], proposed in the context of computer vision, which computes the minimum of the counts at each bin of the two histograms instead of the dot product in the spectrum kernel. Interestingly, this kernel exhibits a well-known relation with optimal transport, see [48]:

Lemma A.1 (Histograms intersection as an optimal match kernel)**.** *The histogram intersection can be cast as the optimization of a match kernel:*

$$K_{hist}(\mathbf{x}, \mathbf{x}') := \sum_{u\in\mathcal{F}} \min(\hat{\mathbf{H}}_u(\mathbf{x}), \hat{\mathbf{H}}_u(\mathbf{x}')) = \max_{P\in U_r(\frac{1}{n}, \frac{1}{n'})} \sum_{i=1}^{n} \sum_{j=1}^{n'} \mathbf{P}_{ij} \delta(\mathbf{x}_i, \mathbf{x'}_j),$$

where U_r is a relaxed Kantorovich coupling constraint

$$U_r\left(\frac{1}{n}, \frac{1}{n'}\right) = \left\{\mathbf{P} \in \mathbb{R}_+^{n\times n'} : \mathbf{P}\mathbf{1}_n \leq \frac{1}{n} \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} \leq \frac{1}{n'}\right\}.$$

Proof. For completeness, a proof can be found in Appendix C. ∎

It is possible to show that the distance induced by K_{hist} is the ℓ_1-norm between two histograms in contrast to the ℓ_2-norm in K_{spectrum}. A similar relation between the intersection kernel and a variation of optimal transport was also studied in [48]. When dealing with discrete feature sets, our kernel K_{OT} differs from K_{hist} in three aspects: (i) the relaxed Kantorovich constraint becomes exact; (ii) we add an entropic penalty term $H(\mathbf{P})$, which brings both flexibility as it interpolates between K_{hist} (when $\varepsilon = 0$) and K_{spectrum} (when ε gets bigger, we maximize the entropy, which is equivalent to summing all the pairs with identical weights as in K_{spectrum}), and computational scalability as well as differentiability thanks to Sinkhorn's algorithm; (iii) we relax δ by a positive definite, differentiable kernel κ allowing mismatches and end-to-end learning [6], *e.g*, a Gaussian kernel.

B Additional details

This section provides additional details on the positional encoding in the OTK, and the reconstruction of the transport plan for Gaussian distributions.

B.1 Position encoding of OTK

We introduced in Subsection 3.4 a kernel form of the OTK integrating positional information. However, its resulting embedding function is not obvious to see as it is not trivial how to approximate the position term with a dot-product. We thus detail here an embedding that works well in practice. After adding the positional term to the OTK, the kernel becomes

$$K_{\mathrm{OT}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \sum_{j=1}^{n'} \mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{x}')_{ij} \kappa'(\mathbf{x}_i, \mathbf{x}'_j),$$

where

$$\kappa'(\mathbf{x}_i, \mathbf{x}'_j) = \kappa(\mathbf{x}_i, \mathbf{x}'_j) \times e^{-\frac{1}{\sigma^2_{\mathrm{pos}}}(i/n - j/n')^2}.$$

Computing the transport plan against a reference measure $\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z})_{ij}$ is similar to the OTK without position encoding, by simply replacing the input similarity matrix $\mathbf{K}$ in (9) with $\mathbf{S} \odot \mathbf{K}$ where $\mathbf{S}$ denotes the matrix with entry $\mathbf{S}_{ij} = e^{-\frac{1}{\sigma^2_{\mathrm{pos}}}(i/n - j/p)^2}$. Whereas, the kernel mapping of κ' is no more φ (or ψ when φ is infinite dimensional) as there is this additional position term. However, we can mimic its effect without adding further dimensions by multiplying elementwisely $\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z})$ with $\mathbf{S}$. This results in the following embedding with position information

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times [\mathbf{P}_{\kappa'}(\mathbf{x}, \mathbf{z}) \odot \mathbf{S}]^\top \varphi(\mathbf{x}).$$

B.2 Reconstruction of the transport for Gaussian distributions

In the case of Gaussian distributions and Monge formulation of optimal transport, the reconstruction formula given in 3.2 is exact. Before we prove this, we introduce the notion of Bures-Wasserstein distance and its well-known relationship to optimal transport. Let x, x' and z be random gaussian vectors in $\mathbb{R}^n$ with zero mean and respective covariance matrices A, B and C. For such distributions, the Bures-Wasserstein distance between A and B, defined as

$$d(A, B) := \left[\mathrm{tr}A + \mathrm{tr}B - 2\mathrm{tr}(A^{1/2} B A^{1/2})^{1/2} \right]^{1/2},$$

coincides with the 2-Wasserstein distance between x and x', $W_2(x, x')$. As a result, Bures-Wasserstein theory (see, *e.g*, [4]) provides a closed form solution to the optimal transport problem cast as

$$W_2(x, x') := \inf_{\gamma \in \Gamma(\mu, \nu)} \left(\int_{\mathbb{R}^n \times \mathbb{R}^n} \|x - y\|^2 d\gamma(x, y) \right)^{1/2},$$

where μ and ν are mass distributions: the minimum is indeed attained in $x' = Tx$, with

$$T = A^{-1} \# B := A^{-1}(AB)^{1/2} = (A^{-1}B^{-1})^{1/2} B.$$

Lemma B.1. *Let x, x' and z be gaussian distributions with covariance matrices A, B and C. We assume that we know the Monge optimal transport mapping $T_{A \to C}$ from x to z and $T_{B \to C}$ from x' to z. Then, the mapping from x to x' can be obtained with*

$$\tilde{T}_{A \to B} := T^{-1}_{B \to C} T_{A \to C}.$$

Proof. If we use the regular transport from x (covariance matrix A) to x' (covariance matrix B) the new covariance matrix is

$$\begin{aligned}\mathbf{E}(T_{A\to B}xx^\top T_{A\to B}^\top) &= T_{A\to B}\mathbf{E}(xx^\top)T_{A\to B}^\top \\ &= T_{A\to B}AT_{A\to B} \\ &= A^{-1}(AB)^{1/2}AA^{-1}(AB)^{1/2} \\ &= B.\end{aligned}$$

Now, transporting x to z then z to x', the reconstructed covariance matrix is

$$\begin{aligned}\mathbf{E}(T_{C\to B}T_{A\to C}xx^\top T_{A\to C}^\top T_{C\to B}^\top) &= T_{C\to B}T_{A\to C}\mathbf{E}(xx^\top)T_{A\to C}^\top T_{C\to B}^\top \\ &= T_{C\to B}T_{A\to C}AT_{A\to C}T_{C\to B} \\ &= T_{C\to B}T_{A\to C}AA^{-1}(AC)^{1/2}T_{C\to B} \\ &= T_{C\to B}A^{-1}ACT_{C\to B} \\ &= T_{C\to B}CT_{C\to B} \\ &= B.\end{aligned}$$

So, transporting x to z before transporting the result to x' is equivalent to directly transporting x to x'. The reconstructed transport yields the same distribution. Since $T_{B\to C}^{-1} = T_{C\to B}$, we can conclude. ■

Remark B.2. *If x and x' have non-zero means, one just needs to use the mapping $\tilde{T}_{A\to B}(x-\mu_x)+\mu_{x'}$.*

C Proofs

C.1 Proof of Lemma A.1

Proof. For any $\mathbf{P} \in U_r\left(\frac{1}{n}, \frac{1}{n'}\right)$, we have

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x}_i, \mathbf{x'}_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n'}\sum_{u\in\mathcal{F}}\mathbf{P}_{ij}\delta(\mathbf{x}_i, u)\delta(\mathbf{x'}_j, u) \\ &= \sum_{u\in\mathcal{F}}\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x}_i, u)\delta(\mathbf{x'}_j, u)}_{:=f_u(\mathbf{x},\mathbf{x}')}.\end{aligned}$$

Then,

$$f_u(\mathbf{x},\mathbf{x}') = \sum_{i=1}^{n}\delta(\mathbf{x}_i,u)\sum_{j=1}^{n'}\mathbf{P}_{ij}\delta(\mathbf{x'}_j,u) \le \sum_{i=1}^{n}\delta(\mathbf{x}_i,u)\underbrace{\sum_{j=1}^{n'}\mathbf{P}_{ij}}_{\le 1/n} \le \mathbf{H}_u(\mathbf{x})/n = \hat{\mathbf{H}}_u(\mathbf{x}),$$

and similarly, $f_u(\mathbf{x},\mathbf{x}') \le \hat{\mathbf{H}}_u(\mathbf{x}')$. Consequently, $f_u(\mathbf{x},\mathbf{x}') \le \min(\hat{\mathbf{H}}_u(\mathbf{x})\hat{\mathbf{H}}_u(\mathbf{x}'))$. Since the index sets $\mathcal{I}_u(\mathbf{x},\mathbf{x}') = \{(i,j)\in[1,n]\times[1,n'] : \mathbf{x}_i = \mathbf{x'}_j = u\}$ are disjoint for different u, we will show that the equality for any $u\in\mathcal{F}$ can be attained with

$$\mathbf{P}_{ij} = \min\left(\frac{1}{n\mathbf{H}_{\mathbf{x}_i}(\mathbf{x}')}, \frac{1}{n'\mathbf{H}_{\mathbf{x'}_j}(\mathbf{x})}\right)\delta(\mathbf{x}_i, \mathbf{x'}_j).$$

First, $\sum_{j=1}^{n'} \mathbf{P}_{ij} = \min(1/n, \mathbf{H}_{\mathbf{x}_i}(\mathbf{x}')/n'\mathbf{H}_{\mathbf{x}_i}(\mathbf{x})) \leq 1/n$ and, similarly, $\sum_{i=1}^{n} \mathbf{P}_{ij} \leq 1/n'$. As a consequence, $\mathbf{P} \in U_r\left(\frac{1}{n}, \frac{1}{n'}\right)$. Moreover,

$$\begin{aligned} f_u(\mathbf{x}, \mathbf{x}') &= \sum_{(i,j)\in\mathcal{I}_u(\mathbf{x},\mathbf{x}')} \mathbf{P}_{ij} \\ &= \mathbf{H}_u(\mathbf{x})\mathbf{H}_u(\mathbf{x}') \min(1/n\mathbf{H}_u(\mathbf{x}'), 1/n'\mathbf{H}_u(\mathbf{x})) \\ &= \min(\hat{\mathbf{H}}_u(\mathbf{x}), \hat{\mathbf{H}}_u(\mathbf{x}')), \end{aligned}$$

so we have an equality case, which concludes the proof. ■

A direct conclusion from this Lemma is that the optimal match problem A.1 defines a positive definite kernel since the histogram intersection kernel is known to be positive definite.

C.2 Proof of Lemma 3.3

Proof. First, since $\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk} = 1$ for any k, we have

$$\begin{aligned} W_2(\mathbf{x}, \mathbf{z})^2 &= \sum_{i=1}^{n}\sum_{k=1}^{p} \mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \|\mathbf{u}\|_2^2, \end{aligned}$$

with $\mathbf{u}$ a vector in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{z}_k)$ for $i = 1, \ldots, n$, $j = 1, \ldots, n'$ and $k = 1, \ldots, p$. We can also rewrite $W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ as an ℓ_2-norm of a vector $\mathbf{v}$ in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{x}'_j)$. Then by Minkowski inequality for the ℓ_2-norm, we have

$$\begin{aligned} |W_2(\mathbf{x}, \mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| &= |\|\mathbf{u}\|_2 - \|\mathbf{v}\|_2| \\ &\leq \|\mathbf{u} - \mathbf{v}\|_2 \\ &= \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}(d_\kappa(\mathbf{x}_i, \mathbf{z}_k) - d_\kappa(\mathbf{x}_i, \mathbf{x}'_j))^2\right)^{1/2} \\ &\leq \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}'_j, \mathbf{z}_k)\right)^{1/2} \\ &= \left(\sum_{k=1}^{p}\sum_{j=1}^{n'} \mathbf{P}(\mathbf{x}', \mathbf{z})_{jk} d_\kappa^2(\mathbf{x}'_j, \mathbf{z}_k)\right)^{1/2} \\ &= W_2(\mathbf{x}', \mathbf{z}), \end{aligned}$$

where the second inequality is the triangle inequality for the distance d_κ. Finally, we have

$$\begin{aligned} &|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \\ \leq &|W_2(\mathbf{x}, \mathbf{x}') - W_2(\mathbf{x}, \mathbf{z})| + |W_2(\mathbf{x}, \mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \\ \leq &W_2(\mathbf{x}', \mathbf{z}) + W_2(\mathbf{x}', \mathbf{z}) \\ = &2W_2(\mathbf{x}', \mathbf{z}), \end{aligned}$$

where the second inequality is the triangle inequality for the 2-Wasserstein distance. By symmetry, we also have $|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')| \leq 2W_2(\mathbf{x}, \mathbf{z})$, which concludes the proof. ■

C.3 Relationship between W_2 and $W_2^{\mathbf{z}}$ for multiple references

Using relation (6) (see Appendix C for details), we can obtain a bound on the error term between W_2 and $W_2^{\mathbf{z}}$ for a data set of m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ and q references $(\mathbf{z}^1, \dots, \mathbf{z}^q)$

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{mq} \sum_{i=1}^{m} \sum_{j=1}^{q} W_2^2(\mathbf{x}^i, \mathbf{z}^j). \tag{10}$$

When $q = 1$, the right-hand term in the inequality is the objective to minimize in the Wasserstein barycenter problem [11], which further explains why we considered it: Once $W_2^{\mathbf{z}}$ is close to the Wasserstein distance W_2, $K_{\mathbf{z}}$ will also be close to K_{OT} thanks to relation (3). We extend here the bound given in Lemma 3.3 in the case of one reference to the multiple-reference case. The approximate 2-Wasserstein distance $W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ thus becomes

$$W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}') := \left\langle \frac{1}{q} \sum_{j=1}^{q} \mathbf{P}_{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \right\rangle^{1/2} = \left(\frac{1}{q} \sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2}.$$

Then by Minkowski inequality for the ℓ_2-norm we have

$$\begin{aligned} |W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}')| &= \left| \left(\frac{1}{q} \sum_{j=1}^{q} W_2(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2} - \left(\frac{1}{q} \sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}')^2 \right)^{1/2} \right| \\ &\leq \left(\frac{1}{q} \sum_{j=1}^{q} (W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^j}(\mathbf{x}, \mathbf{x}'))^2 \right)^{1/2}, \end{aligned}$$

and by Lemma 3.3 we have

$$|W_2(\mathbf{x}, \mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}, \mathbf{x}')| \leq \left(\frac{4}{q} \sum_{j=1}^{q} \min(W_2(\mathbf{x}, \mathbf{z}^j), W_2(\mathbf{x}', \mathbf{z}^j))^2 \right)^{1/2}.$$

Finally the approximation error in terms of Frobenius is bounded by

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{mq} \sum_{i=1}^{m} \sum_{j=1}^{q} W_2^2(\mathbf{x}^i, \mathbf{z}^j).$$

In particular, when $q = 1$ that is the case of single reference, we have

$$\mathcal{E}^2 \leq \frac{4}{m} \sum_{i=1}^{m} W_2^2(\mathbf{x}^i, \mathbf{z}),$$

where the right term equals to the objective of the Wasserstein barycenter problem, which justifies the choice of $\mathbf{z}$ when learning without supervision.

D Additional experimental results

In this section, we provide details on our experimental protocol for reproducibility, as well as additional experimental results. The results are generally averaged over different runs, and the uncertainty is represented with the standard deviation.

Table 6: Accuracies obtained by computing the kernel matrices on 5000 samples of CIFAR-10 for various feature size: [(patch × patch), embedding dimension]. Metric: accuracy on validation set.

Dataset	Mean pooling	Linear kernel	K_{OT}	$K_{\mathbf{z}}$	K_{OT}+ Pos. enc.	$K_{\mathbf{z}}$+ Pos. enc.
(3 × 3), 256	0.584	0.649	0.619	0.61	**0.661**	0.652
(3 × 3), 8192	0.636	0.690	0.652	0.65	0.693	**0.694**

Table 7: Hyperparameter search range for CIFAR-10

Hyperparameter	Search range
Entropic regularization ε	$[1.0; 0.1; 0.01; 0.001]$
Position encoding bandwidth σ_{pos}	$[0.5; 0.6; 0.7; 0.8; 0.9; 1.0]$

D.1 Experiments on Kernel Matrices.

Here, we study the optimal transport kernel K_{OT} (2) and its surrogate $K_{\mathbf{z}}$ (4) which exhibit interesting properties. Although our embedding $\Phi_{\mathbf{z}}$ is scalable, the exact kernel require the computation of Gram matrices. Therefore, 5000 samples only of CIFAR-10 (images with 32×32 pixels) are encoded without supervision using a two-layer convolutional kernel network [24]. The resulting features are 3×3 patches living in $\mathbb{R}^d$ with $d = 256$ or 8192. Since the features are already the output of a Gaussian Nyström embedding, the intermediate kernel κ is linear, which means that K_{OT} and $K_{\mathbf{z}}$ aggregate existing features linearly given the computed weight matrix $\mathbf{P}$. In that sense, we can say that our kernels work as an adaptive pooling. We therefore compare it to kernel matrices corresponding to mean pooling and no pooling at all (linear kernel). A linear classifier is trained from this matrices. Although we cannot prove that K_{OT} is positive definite, the classifier trained on the kernel matrix converges when ε is not too small. The results can be seen on Table 6. Without positional information, our kernels do better than Mean pooling. When the positions are encoded, the Linear kernel is also outperformed. Note that including positions in Mean pooling and Linear kernel means interpolating between these two kernels: in the Linear kernel, only patches with same index are compared while in the Mean pooling, all patches are compared. All interpolations did worse than the Linear kernel.

D.2 CIFAR-10

We build our OTK on top of the state-of-the-art unsupervised features for CIFAR-10, extracted from a 2-layer CKN [26, 24] model with kernel sizes equal to 3 and 3, and Gaussian pooling size equal to 2 and 1. We consider the following configurations of the number of filters at each layer, to simulate different input dimensions for OTKs

- 64 filters at first and 256 at second layer, which yields a $16 \times 16 \times 256$ representation for each image;
- 256 filters at first and 1024 at second layer, which yields a $16 \times 16 \times 1024$ representation for each image.

We feed to OTKs the output features of a CKN model, which is already a kernel embedding. κ in OTKs will therefore be a linear kernel. The OTK embedding is learned with one reference using K-means method described in Section 3 and compared to several classical pooling baselines, including the original CKN's Gaussian pooling with pooling size equal to 6. The hyperparameters are entropic regularization ε and bandwidth for position encoding σ_{pos}. Their search grids are shown in Table 7. The results are shown in Table 8. We notice that the position encoding is crucial to this task, and substantially improves the performance of its counterpart without it.

Table 8: Classification results for CIFAR-10. We consider here OTKs with one reference with different number of supports, learned with K-means. The embeddings are computed with or without position encoding (PE).

Method	Nb. supports	$16 \times 16 \times 256$	$16 \times 16 \times 1024$
Flatten		73.1	76.1
Mean pooling		64.9	73.4
Gaussian pooling [24]		77.5	82.0
OTK	9	75.6	79.3
OTK (with PE)		78.0	82.2
OTK	64	77.9	80.1
OTK (with PE)		81.4	83.2
OTK	144	78.4	80.7
OTK (with PE)		81.8	83.4

Table 9: Hyperparameter search grid for SCOP 1.75

Hyperparameter	Search range
ε for Sinkhorn	$[1.0; 0.5; 0.1; 0.05; 0.01]$
λ for classifier (unsupervised OTKs)	$1/2^{\text{range}(5,20)}$
λ for classifier (supervised OTKs)	[1e-6;1e-5;1e-4;1e-3]

D.3 Protein fold recognition

Dataset description. Our protein fold recognition experiments consider the Structural Classification Of Proteins (SCOP) version 1.75 and 2.06. We follow the data preprocessing protocols in [15], which yields a training and validation set composed of 14699 and 2013 sequences from SCOP 1.75, and a test set of 2533 sequences from SCOP 2.06. The resulting protein sequences belong to 1195 different folds, thus the problem is formulated as a multi-classification task. The input sequence is represented as a 45-dimensional vector at each amino acid. The vector consists of a 20-dimensional one-hot encoding of the sequence, a 20-dimensional position-specific scoring matrix (PSSM) representing the profile of amino acids, a 3-class secondary structure represented by a one-hot vector and a 2-class solvent accessibility. The lengths of the sequences are varying from tens to thousands.

Models setting and hyperparameters. We consider here the one-layer models followed by a global mean pooling for the baseline methods CKN [6] and RKN [7]. We build our OTK on top of the one-layer CKN model, where κ can be seen as a Gaussian kernel on the k-mers in sequences. The only difference between our model and CKN is thus the pooling operation, which is given by our embedding introduced in Section 3. The bandwidth parameter of the Gaussian kernel κ on k-mers is fixed to 0.6 for unsupervised models and 0.5 for supervised models, the same as used in CKN which were selected by the accuracy on the validation set. The filter size k is fixed to 10 and different numbers of anchor points in Nyström for κ are considered in the experiments. The other hyperparameters for OTKs are the entropic regularization parameter ε, the number of supports in a reference p, the number of references q, the number of iterations for Sinkhorn's algorithm and the regularization parameter λ in the linear classifier. The search grid for ε and λ is shown in Table 9 and they are selected by the accuracy on validation set. ε plays an important role in the performance and is observed to be stable for the same dataset. For this dataset, it is selected to be 0.5 for all the unsupervised and supervised models. The effect of other hyperparameters will be discussed below.

Learning unsupervised OTKs. The kernel embedding φ, which is infinite dimensional for the Gaussian kernel, is approximated with the Nyström method using K-means on 300000 k-mers extracted from the same training set as in [7]. The reference measures are learned by using either K-means or Wasserstein to update

Table 10: Classification accuracy (top 1/5/10) results of unsupervised OTKs for SCOP 1.75. We show the results for different combinations of (number of references q × number of supports p). The reference measures $\mathbf{z}$ are learned with either K-means or Wasserstein barycenter for updating centroids.

Nb. filters	Method	q	Embedding size ($q \times p$)			
			10	50	100	200
128	K-means	1	76.5/91.5/94.4	77.5/91.7/94.5	79.4/92.4/94.9	78.7/92.1/95.1
		5	72.8/89.9/93.7	77.8/91.7/94.6	78.6/91.9/94.6	78.1/92.1/94.7
		10	62.7/85.8/91.1	76.5/91.0/94.2	78.1/92.2/94.9	78.6/92.2/94.7
	Wass. bary.	1	64.0/85.9/91.5	71.6/88.9/93.2	77.2/91.4/94.2	77.5/91.9/94.8
		5	70.5/89.1/93.0	76.6/91.3/94.4	78.4/91.7/94.3	77.1/91.9/94.7
		10	63.0/85.7/91.0	75.9/91.4/94.3	77.5/91.9/94.6	77.7/92.0/94.7
1024	K-means	1	84.4/95.0/96.6	84.6/95.0/97.0	85.7/95.3/96.7	85.4/95.2/96.7
		5	81.1/94.0/96.2	84.9/94.8/96.8	84.7/94.4/96.7	85.2/95.0/96.7
		10	79.8/93.5/95.9	83.1/94.6/96.6	84.4/94.7/96.7	84.8/94.9/96.7

Table 11: Classification accuracy (top 1/5/10) of supervised models for SCOP 1.75. The accuracies obtained by averaging 10 different runs. We show the results of using either one reference with 50 supports or 5 references with 10 supports. Here DeepSF is a 10-layer CNN model.

Method	Top 1/5/10 accuracy on SCOP 2.06	
PSI-BLAST [15]	84.53/86.48/87.34	
DeepSF [15]	73.00/90.25/94.51	
Number of filters	128	512
CKN [6]	76.30±0.70/92.17±0.16/95.27±0.17	84.11±0.11/94.29±0.20/96.36±0.13
RKN [7]	77.82±0.35/92.89±0.19/95.51±0.20	85.29±0.27/94.95±0.15/96.54±0.12
Ours		
OTK ($\Phi_{\mathbf{z}}$ 1 × 50)	82.83±0.41/93.89±0.33/96.23±0.12	88.40±0.22/95.76±0.13/97.10±0.15
OTK ($\Phi_{\mathbf{z}}$ 5 × 10)	**84.68±0.50/94.68±0.29/96.49±0.18**	**88.66±0.25/95.90±0.15/97.33±0.14**

centroids in 2-Wasserstein K-means on 3000 subsampled sequences for RAM-saving reason. We evaluate OTKs on top of features extracted from CKNs of different dimensions, representing the number of anchor points used to approximate κ. The number of iterations for Sinkhorn is fixed to 100 to ensure the convergence. The results for different combinations of q and p are provided in Table 10. Increasing the number of supports p can improve the performance and then saturate it when p is too large. On the other hand, increasing the number of references while keeping the embedding dimension (*i.e.* $p \times q$) constant is not significantly helpful in this unsupervised setting. We also notice that Wasserstein Barycenter for learning the references does not outperform K-means, while the latter is faster in terms of computation.

Learning supervised OTKs. The supervised OTKs are initialized with the unsupervised method and then trained in an alternating fashion which was also used for CKN: we use an Adam algorithm to update anchor points in Nyström and reference measures $\mathbf{z}$, and the L-BFGS algorithm to optimize the classifier. The learning rate for Adam is initialized with 0.01 and halved as long as there is no decrease of the validation loss for 5 successive epochs. In practice, we notice that using a small number of Sinkhorn iterations can achieve similar performance to a large number of iteration, while being much faster to compute. We thus fix it to 10 throughout the experiments. The accuracy results are obtained by averaging on 10 runs with different seeds following the setting in [7]. The results are shown in Table 11 with error bars. The effect of the number of supports q is similar to the unsupervised case, while increasing the number of references can indeed improve performance.

Table 12: Model architecture for DeepSEA dataset.

Model architecture
Conv1d(in channels=4, out channels=d, kernel size=16) + ReLU
OTKLayer(in channels=d, supports=64, references=1, $\varepsilon = 1.0$, PE=True, $\sigma_{\text{pos}} = 0.1$)
Linear(in channels=d, out channels=d) + ReLU
Dropout(0.4)
Linear(in channels=$d \times 64$, out channels=919) + ReLU
Linear(in channels=919, out channels=919)

D.4 Detection of chromatin profiles

Dataset description. Predicting the functional effects of noncoding variants from only genomic sequences is a central task in human genetics. A fundamental step for this task is to simultaneously predict large-scale chromatin features from DNA sequences [47]. We consider here the DeepSEA dataset, which consists in simultaneously predicting 919 chromatin profiles including 690 transcription factor (TF) binding profiles for 160 different TFs, 125 DNase I sensitivity profiles and 104 histone-mark profiles. In total, there are 4.4 million, 8000 and 455024 samples for training, validation and test. Each sample consists of a 1000-bp DNA sequence from the human GRCh37 reference. Each sequence is represented as a 1000×4 binary matrix using one-hot encoding on DNA characters. The dataset is available at `http://deepsea.princeton.edu/media/code/deepsea_train_bundle.v0.9.tar.gz`. Note that the labels for each profile are very imbalanced in this task with few positive samples. For this reason, learning unsupervised models could be intractable as they may require an extremely large number of parameters if junk or redundant sequences cannot be filtered out.

Model architecture and hyperparameters. For the above reason and fair comparison, we use here the supervised OTK as a module in Deep NNs. The architecture of our model is shown in Table 12. We use an Adam optimizer with initial learning rate equal to 0.01 and halved at epoch 1, 4, 8 for 15 epochs in total. The number of iterations for Sinkhorn is fixed to 30. The whole training process takes about 30 hours on a single GTX2080TI GPU. The dropout rate is selected to be 0.4 from the grid $[0.1; 0.2; 0.3; 0.4; 0.5]$ and the weight decay is 1e-06, the same as [47]. The σ_{pos} for position encoding is selected to be 0.1, by the validation accuracy on the grid $[0.05; 0.1; 0.2; 0.3; 0.4; 0.5]$. The checkpoint with the best validation accuracy is used to evaluate on the test set. Area under ROC (auROC) and area under precision curve (auPRC), averaged over 919 chromatin profiles, are used to measure the performance. The hidden size d is chosen to be either 1024 or 1536.

Results and importance of position encoding. We compare our model to the state-of-the-art CNN model DeepSEA [47] with 3 convolutional layers. Our model outperforms DeepSEA, while requiring fewer layers. The positional information is known to be important in this task. To show the efficacy of our position encoding, we compare it to the sinusoidal encoding used in the original transformer [40]. We observe that our encoding with properly tuned σ_{pos} requires fewer layers, while being interpretable from a kernel point of view. We also find that larger hidden size d performs better, as shown in Table 13. ROC and PR curves for all the chromatin profiles and stratified by transcription factors, DNase I-hypersensitive sites and histone-marks can also be found in Figure 2.

D.5 SST-2

Dataset description. The data set contains 67349 training samples and 872 validation samples and can be found at `https://gluebenchmark.com/tasks`. The test set contains 1821 samples for which the predictions need to be submitted on the GLUE leaderboard, with limited number of submissions. As a consequence, our training and validation set are extracted from the original training set (80% of the original training set is

Table 13: Results for prediction of chromatin profiles on the DeepSEA dataset. The metrics are area under ROC (auROC) and area under PR curve (auPRC), averaged over 919 chromatin profiles. The accuracies are averaged from 10 different runs. Armed with the positional encoding (PE) described in Section 3, the OTK outperforms the state-of-the-art model and an OTK with the PE proposed in [40].

Method	DeepSEA [47]	OTK	OTK ($d = 1024$)	OTK ($d = 1536$)
Position encoding	-	Sinusoidal [40]	Ours	Ours
auROC	0.933	0.917	0.935	**0.936**
auPRC	0.342	0.311	0.354	**0.360**

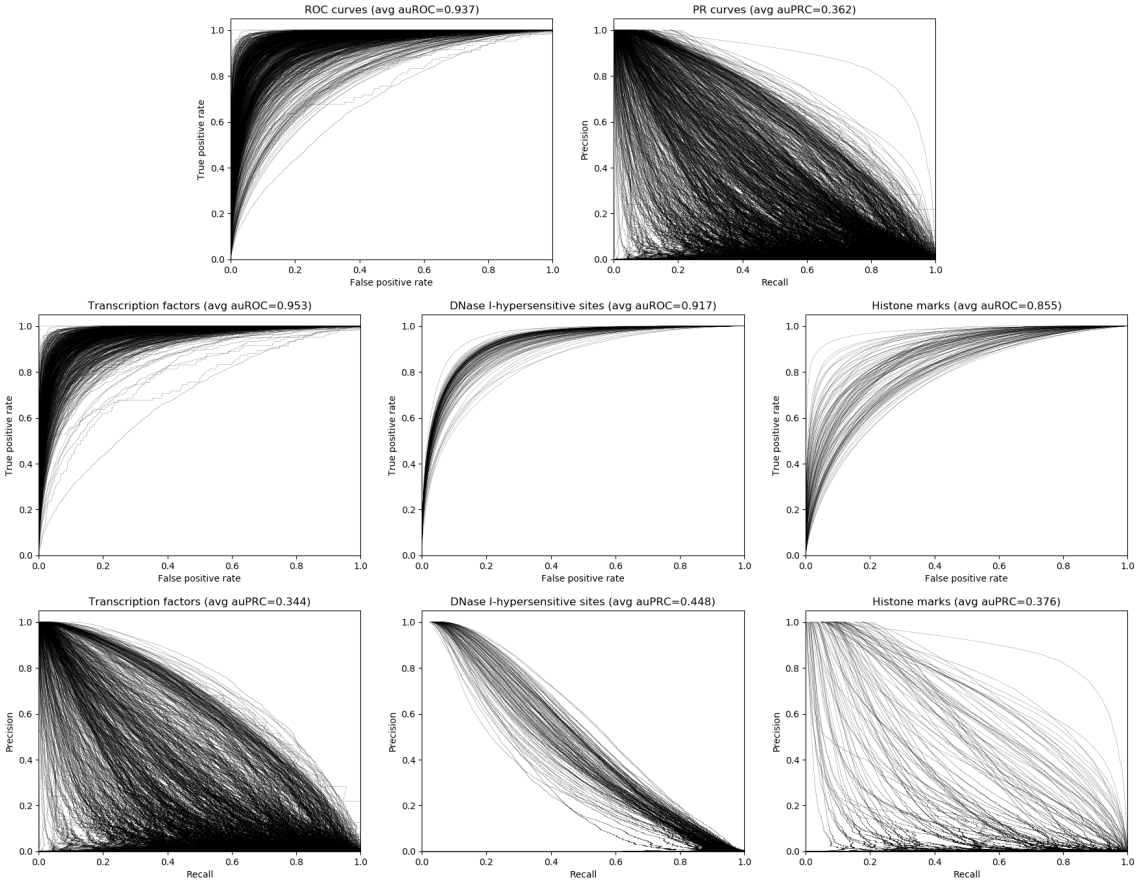

Figure 2: ROC and PR curves for all the chromatin profiles (first row) and stratified by transcription factors (left column), DNase I-hypersensitive sites (middle column) and histone-marks (right column). The profiles with positive samples fewer than 50 on the test set are not taken into account.

Table 14: Accuracies on standard validation set for SST-2 with unsupervised OTK features depending on the number of references and supports. The references were computed using K-means on samples for multiple references and K-means on patches for multiple supports. The size of the input BERT features is (length × dimension). The accuracies are averaged from 10 different runs.

BERT Input Feature Size	(30 × 768)		(66 × 768)	
Features	Pre-trained	Fine-tuned	Pre-trained	Fine-tuned
[CLS]	84.6±0.3	90.3±0.1	86.0±0.2	**92.8±0.1**
Flatten	84.9±0.4	**91.0±0.1**	85.2±0.3	92.5±0.1
Mean pooling	85.3±0.3	90.8±0.1	85.4±0.3	92.6±0.2
OTK (1 × 3)	85.5±0.1	90.9±0.1	86.5±0.1	92.6±0.1
OTK (1 × 10)	85.1±0.4	90.9±0.1	85.9±0.3	92.6±0.1
OTK (1 × 30)	86.3±0.3	90.8±0.1	86.6±0.5	92.6±0.1
OTK (1 × 100)	85.7±0.7	90.9±0.1	86.6±0.1	92.7±0.1
OTK (1 × 300)	**86.8±0.3**	90.9±0.1	**87.2±0.1**	92.7±0.1

used for our training set and the remaining 20% is used for our validation set), and we report accuracies on the standard validation set, used as a test set. The reviews are padded with zeros when their length is shorter than the chosen sequence length (we choose 30 and 66, the latter being the maximum review length in the data set) and the BERT implementation requires to add special tokens [CLS] and [SEP] at the beginning and the end of each review.

Model architecture and hyperparameters. In most transformers such as BERT, the embedding associated to the token [CLS] is used for classification and can be seen in some sense as an embedding of the review adapted to the task. The features we used are the word features provided by the BERT base-uncased version, available at `https://huggingface.co/transformers/pretrained_models.html`. For this version, the dimension of the word features is 768. In the unsupervised case, the word embeddings of the reviews are kept as is, *i.e* we do not embed it using a Gaussian kernel. In this setting, the OTK linearly recombines the word features based on the transport plan. The resulting features are used to train a large-scale linear classifier using the Cyanure library [25]. In the supervised case, the OTK uses a Gaussian Nyström embedding with varying number of filters before the pooling layer, and the parameters of the two layers, $\mathbf{w}$ and $\mathbf{z}$, are optimized end-to-end. In this case, we have to tune the bandwidth of the Gaussian kernel as well as the learning rate. The classifier is here a fully-connected layer. In both case, we tune the entropic regularization parameter of optimal transport and the regularization parameter (or weight decay) of the classifier so as to get the best accuracy on the standard validation set, which is our test set. The parameters in the search grid are summed up in Table 15. The best entropic regularization and Gaussian kernel bandwidth are typically ans respectively 3.0 and 0.5 for this data set. In BERT models, the positional information is integrated in the initial word embeddings. As a consequence, we do not use our own positional encoding. The supervised training process takes between half an hour for smaller models (typically 128 filters in $\mathbf{w}$ and 3 supports in $\mathbf{z}$) and a few hours for larger models (256 filters and 100 supports) on a single GTX2080TI GPU.

Results and discussion. As explained in Section 5, our unsupervised OTK improves the BERT pre-trained features while still using a simple linear model as shown in Table 14, and its supervised counterpart enables to get even closer to the state-of-the art (for the BERT base-uncased model) accuracy, which is usually obtained after fine-tuning of the BERT model on the whole data set. This can be seen in Table 16.

Table 15: Hyperparameter search grid for SST-2.

Hyperparameter	Search range
Entropic regularization ε	$[3.0; 1.0; 0.5]$
λ for classifier (unsupervised OTKs)	$10^{\text{range}(-10,1)}$
λ for classifier (supervised OTKs)	[1e-4;1e-3;1e-2]
Gaussian kernel bandwidth (supervised OTKs)	$[0.5; 1.0; 1.5]$
Learning rate (supervised OTKs)	$[0.1; 0.01; 0.001]$

Table 16: Accuracies on standard validation set for SST-2 with supervised OTK features from pre-trained BERT (30×768) depending on the number of supports in the reference. The accuracies are averaged from 10 different runs, and 30 Sinkhorn iterations were used.

Number of supports p		3	10	30
Nyström filters	128	87.59	87.59	87.53
	256	87.45	87.44	87.50
	512	87.27	87.16	87.29