Predicting Transcription Factor Binding Sites with Convolutional Kernel Networks

Dexiong Chen, Laurent Jacob, Julien Mairal

Predicting Transcription Factor Binding Sites with Convolutional Kernel Networks

Dexiong Chen[*†] Laurent Jacob[‡] Julien Mairal[*]

Abstract

The growing amount of biological sequences available makes it possible to learn genotype-phenotype relationships from data with increasingly high accuracy. By exploiting large sets of sequences with known phenotypes, machine learning methods can be used to build functions that predict the phenotype of new, unannotated sequences. In particular, deep neural networks have recently obtained good performances on such prediction tasks, but are notoriously difficult to analyze or interpret. Here, we introduce a hybrid approach between kernel methods and convolutional neural networks for sequences, which retains the ability of neural networks to learn good representations for a learning problem at hand, while defining a well characterized Hilbert space to describe prediction functions. Our method outperforms state-of-the-art convolutional neural networks on a transcription factor binding prediction task while being much faster to train and yielding more stable and interpretable results.

Source code is freely available at `https://gitlab.inria.fr/dchen/CKN-seq`.

[*]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France

[†]Email: dexiong.chen@inria.fr

[‡]Univ. Lyon, Université Lyon 1, CNRS, Laboratoire de Biométrie et Biologie Evolutive UMR 5558, Lyon, France

1 Introduction

Understanding and predicting the relationship between biological sequences and their phenotypes is an important problem in molecular biology, with potential applications in both basic and applied science. Accordingly, machine learning techniques have been developed over the past decade and a half to exploit the growing amount of phenotypic sequences in automatic annotation tools. Typical applications include classifying protein domains into superfamilies [24, 32], predicting whether a DNA or RNA sequence binds to a protein [1], its splicing outcome [15], or its chromatin accessibility [16], predicting the resistance of a bacterial strain to a drug [11], or denoising a ChIP-seq signal [18].

Underlying the problem of the genotype-phenotype relationship is the question of biological sequence representation: whether one wants to understand why a replication origin is more active than another one [31] or to predict the function of a protein [5], one necessarily relies on a particular way to describe sequences, which should allow a good discrimination among biological functions in the latter problem, and contain causal elements shedding light on the mechanism of replication origins for the former one. Early methods for studying the genotype-phenotype relationship relied on pre-determined fingerprints [3] or hidden Markov models [19]. A related line of work focused on position weight matrices (PWMs) to characterize a set of sequences [42]. Most of the methods to build PWMs are unsupervised: they identify recurrent patterns found in a group of samples, *e.g.*, known binders of a transcription factor, and are easy to interpret. Yet, a few methods specifically build motifs that discriminate between classes using supervision [4]. Recent methods based on convolutional neural networks such as [1] offer heuristics to interpret their filters as PWMs.

Another line of work for genotype-phenotype modeling is based on kernel methods [34]. Biological sequences are typically embedded into a space of very large of infinite dimension and represented as a rich set of descriptors, based for instance on the Fisher score [14], k-mer spectrum up to some mismatches [24], or local alignments [32]. By using the so-called kernel trick, these huge-dimensional descriptors never need to be explicitly computed as long as there is an efficient way to compute the inner-products between pairs of such vectors. Traditional kernel methods are however unable to take full advantage of the large amount of sequence data that recently became available for two main reasons. First, they are poorly scalable, requiring the computation of an $n \times n$ kernel matrix. Second and more important, most kernels use fixed representations of data, as opposed to being optimized for a specific task. In other words, learning and data representation are decoupled.

By contrast, convolutional neural networks are much more scalable and optimize the data representation for a particular learning problem, which allow to build different data descriptors for sequences with different phenotypes. When enough data is available and the problem is suited to convolutional networks, these methods can lead to substantial improvements in learning tasks. This has been the case for computer vision [22], natural language processing [10], audio signals [23], and more recently biological sequences [46, 1, 2]. Deep networks however suffer from a few known limitations. An important one is their lack of interpretability [9]. More generally, the set of functions described by the network is only characterized by its algorithmic construction: no intensional definition is available, making both its analysis and interpretation difficult. In particular, correctly regularizing neural networks to avoid overfitting is still an open issue and involves various heuristics such as dropout [40] and weight decay [13, 20]. In addition, training deep neural networks requires some care and hyper-parameter tuning. Due to non-convexity of the objective, a stochastic gradient descent algorithm is used to train the model, which requires several hyper-parameters to be specified such as a learning rate or a way to initialize the model. Because optimal hyper-parameters are dataset-dependent an exploration is necessary to achieve good results automatically without user

interaction, sometimes resulting in a large increase of the training time.

In this paper, we adopt a hybrid approach between kernel methods and deep neural networks, by adapting the convolutional kernel network (CKN) model originally developed in [27, 26] for image data. Accordingly, our first contribution is to introduce a method which efficiently learns a relevant representation of biological sequences for a learning problem at hand, while enjoying the same well-defined Hilbert space construction as kernel methods. We show in particular how this construction helps producing a more principled interpretation of the trained model. As a second contribution, we show that our CKN is well suited to a faster optimization scheme. Concretely, our method dubbed CKN-Seq achieves similar or better performance than DeepBind [1] on a transcription factor (TF) binding prediction task, while being less sensitive to the choice of hyper-parameters. This allows us to run all our experiments with default hyper-parameter values without user interaction or model calibration, resulting in a significant speed-up. Finally, we propose a multitask learning scheme for CKNs, which further improves the prediction performance against individual models.

2 Methods

In this section, we introduce our approach to prediction of transcription factor binding sites. We first review classical supervised learning paradigms such as deep neural networks and kernel methods, before introducing our convolutional kernel network (CKN) and its multi-task version. We conclude this section by discussing how a trained CKN can be interpreted and visualized.

2.1 Supervised Learning

Let us consider n samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in a set $\mathcal{X}$ of possibly variable-length biological sequences. The sequences are assumed to be over an alphabet $\mathcal{A}$. Each sequence $\mathbf{x}_i$ is associated to a measurement y_i in $\mathcal{Y}$ denoting some biological property of the sequence. $\mathcal{Y}$ may be binary labels $\{-1, 1\}$ (*e.g.*, whether the sequence binds a particular transcription factor or not) or $\mathbb{R}$ for continuous traits (*e.g.*, the expression of a gene). The goal of supervised learning is to use these n examples $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,n}$ to build a function $f : \mathcal{X} \mapsto \mathcal{Y}$ which accurately predicts the label of a new, unobserved sequence, typically by minimizing the following objective:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda \Omega(f), \tag{1}$$

where L is a loss function measuring the fit between the true label y_i and the prediction $f(\mathbf{x}_i)$, and Ω measures the regularity of f. $\mathcal{F}$ is a set of candidate functions over which the optimization is performed. Both convolutional neural networks and kernel methods can be thought of as ways to design this set.

Convolutional Neural Networks. In artificial neural networks, the functions in $\mathcal{F}$ perform a sequence of linear and nonlinear operations that are interleaved in a multilayer fashion. More specifically, the convolutional neural network DeepBind [1] represents the four DNA characters as the vectors $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$, respectively, such that an input sequence $\mathbf{x}$ of length m is represented as an $\mathbb{R}^{4 \times m}$ matrix. DeepBind then produces an intermediate representation obtained by one-dimensional convolution of the full sequence $\mathbf{x}$ with p convolution filters, followed by a pointwise non-linear function and a max pooling operation, yielding a representation $\tilde{\mathbf{x}}$ in $\mathbb{R}^p$ of the sequence. A final linear prediction layer is applied to the $\tilde{\mathbf{x}}$. The optimization in (1) acts on

both the weights of this linear function and the convolution filters. DeepBind therefore amounts to learning an optimal representation $\tilde{\mathbf{x}}$ and a linear prediction function over this representation.

DeepBind additionally modifies the objective function (1) to enforce an invariance to reverse complementation of $\mathbf{x}$. The loss term is replaced by $L\left(y_i, \max\left(f(\mathbf{x}_i), f(\bar{\mathbf{x}}_i)\right)\right)$ where $\bar{\mathbf{x}}$ denotes the reverse complement of $\mathbf{x}$. Using this formulation is reported to improve the prediction performance in [1]. Other variants have been then considered, by using a fully connected layer that allows mixing information from the two DNA strands [37], or by considering several hidden layers instead of a single one [45]. Overall, across several variants, the performance of DeepBind with a single hidden layer turned out to be the best on average on ChIP-seq experiments from ENCODE [45].

Kernel Methods. Similar to DeepBind, the main principle of kernel methods is to implicitly map each training point $\mathbf{x}_i$ to a feature space in which predictive functions are simply linear forms. For kernel methods, these feature spaces are vector spaces of high (or even infinite) dimension. This is achieved indirectly, by defining a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which acts as a similarity measure between input data. When the kernel function is symmetric and positive definite, a classical result [35] states that there exists a Hilbert space $\mathcal{F}$ of functions from $\mathcal{X}$ to $\mathbb{R}$, called reproducing kernel Hilbert space (RKHS), along with a mapping $\varphi : \mathcal{X} \to \mathcal{F}$, such that $\langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{F}} = K(\mathbf{x}, \mathbf{x}')$ for all $(\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2$. In other words, there exists a mapping of sequences into a Hilbert space, such that the kernel between any two sequences is equal to the inner-product between the sequences mapped into the Hilbert space. For vectorial data living already in a Euclidean or Hilbert space, classical positive definite kernels are the Gaussian and exponential kernels, respectively defined as $K(\mathbf{x}, \mathbf{x}') = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{x}'\|^2}$ and $K(\mathbf{x}, \mathbf{x}') = e^{\alpha\langle \mathbf{x}, \mathbf{x}' \rangle}$ for $\sigma, \alpha > 0$. A large number of kernels have also been specifically designed for biological sequences (see [6] and references therein).

In the context of supervised learning, training points $\mathbf{x}_i$ can be mapped into $\varphi(\mathbf{x}_i)$ in $\mathcal{F}$, and it becomes natural to look for a prediction function f in $\mathcal{F}$ defined as a linear form $f(\mathbf{x}) = \langle f, \varphi(\mathbf{x}) \rangle_{\mathcal{F}}$. Conveniently, the representer theorem [33] provides that even though f may live in a space of infinite dimension, the optimal solution $f^\star$ of (1) may be parametrized by n scalar values and (1) is equivalent to an optimization problem in $\mathbb{R}^n$, which is convex if L is convex. This equivalent problem only depends on the mapped training points $\varphi(\mathbf{x}_i)$ through inner-products, and can therefore be easily solved as long as the kernel evaluation is fast enough, without ever needing to explicitly compute, store, or manipulate the $\varphi(\mathbf{x}_i)$.

Kernel methods have several assets: (i) they are generic and can be directly applied to any type of data – *e.g.*, sequences or graphs – as soon as a relevant positive definite similarity measure is available; (ii) they are underlied by well defined Hilbert spaces, making their analysis easier. In particular, a natural regularization function $\Omega(f)$ is the squared the norm $\|f\|_{\mathcal{F}}^2$, which allow to control the smoothness of f according to the geometry induced by the kernel.

As alluded to in the introduction, naive implementations of kernel methods lack scalability since they require in general to manipulate the $n \times n$ kernel matrix, which is infeasible if n is large. A typical workaround is the Nyström approximation [44, 39], which builds an explicit low dimension mapping $\psi : \mathcal{X} \to \mathbb{R}^q$ for a reasonably small q, such that $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle_{\mathbb{R}^q} \simeq K(\mathbf{x}, \mathbf{x}')$.

2.2 Convolutional Kernel Networks for Sequences

We now introduce CKNs and discuss how they can lead to learning a similar representation as DeepBind while building a hypothesis space $\mathcal{F}$ which is a RKHS.

Fixed, single-layer construction. CKNs implicitly build $\mathcal{F}$ by borrowing ingredients from the success of convolutional neural networks, namely their ability to model the local stationarity

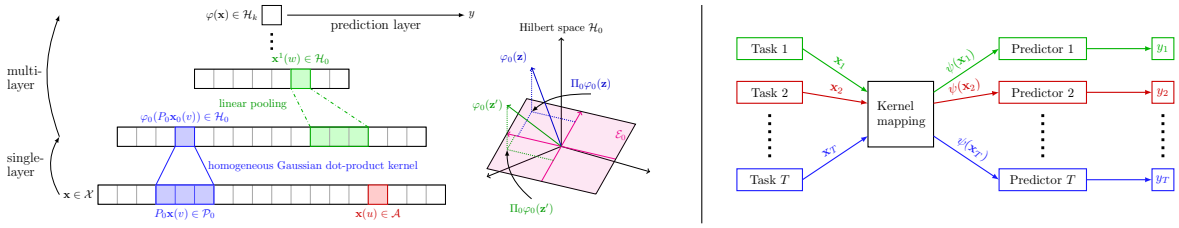


Figure 1: Construction scheme of CKN-seq (left) and its multitask learning scheme (right)

of signals at multiple scales. Even though it is unclear that biological sequence have multiscale properties, modeling local stationarity, on the other hand, turned out to be very important for prediction. From a kernel method point of view, modeling local stationarity requires defining a positive definite kernel K_0 that operates on small local subsequences of length l, which we will call "patch", following image processing terminology. With K_0 comes a reproducing kernel Hilbert space $\mathcal{H}_0$ and a mapping φ_0 such that every patch can be mapped to a point in $\mathcal{H}_0$, as illustrated in Figure 1. The construction provides us an intermediate representation of an input biological sequence as a sequence of points in $\mathcal{H}_0$, each of which carries information from a local neighborhood. In a single-layer model, the intermediate representation is then pooled into a single point in $\mathcal{H}_0$, by computing the barycenter of the patch representations in $\mathcal{H}_0$. We denote by $\Phi(\mathbf{x})$ such a representation in $\mathcal{H}_0$ of a sequence $\mathbf{x}$. As in classical kernel methods, we may now consider the supervised learning formulations (1) with $\mathcal{F} = \mathcal{H}_0$ and look for a prediction function f which is a linear form: $f(\mathbf{x}) = \langle f, \Phi(\mathbf{x}) \rangle_{\mathcal{H}_0}$, while the regularization function is $\Omega(f) = \|f\|^2_{\mathcal{H}_0}$.

While any kernel for sequences could be used, we have focused so far on simpler choices for K_0. Following DeepBind, every character is mapped to a binary vector in $\mathbb{R}^4$, providing a Euclidean structure to the patches. K_0 may then be defined as the Gaussian or exponential kernel over this representation. The main reason for this choice is that [26] defines learning algorithms that rely on such kernels, and which can operate on large-scale data. In the future, we plan to also investigate the use of traditional kernels for biological sequences (see, *e.g.*, [6]).

Learning the kernel. The previous construction decouples learning from data representation, and therefore seems to suffer from the same limitations as kernel methods. However, CKNs propose a mechanism to circumvent this limitation based on a variant of the Nyström approximation. The approximation consists of learning a linear subspace $\mathcal{E}_0$ of finite dimension p_0 within the Hilbert space $\mathcal{H}_0$, as illustrated on the right of Figure 1. This approximation is defined by a set of anchor points $(\mathbf{z}_i)_{i=1,\dots,p_0}$ in $\mathbb{R}^{4\times l}$: $\mathcal{E}_0 := \text{span}(\varphi_0(\mathbf{z}_1), \dots, \varphi_0(\mathbf{z}_{p_0}))$. Every patch $\mathbf{z}$ –or rather, its mapping $\varphi_0(\mathbf{z})$ into $\mathcal{H}_0$– can then be projected into $\mathcal{E}_0$, yielding a representation of the patch by an $\mathbb{R}^{p_0}$ vector $\psi_0(\mathbf{z}) := \mathbf{K}_{ZZ}^{-\frac{1}{2}} \mathbf{K}_Z(\mathbf{z})$, where $\mathbf{K}_{ZZ}^{-\frac{1}{2}}$ is the inverse (or pseudo inverse) square root of the $p_0 \times p_0$ Gram matrix $[K_0(\mathbf{z}_i, \mathbf{z}_j)]_{ij}$ and $\mathbf{K}_Z(\mathbf{z}) = (K_0(\mathbf{z}_1, \mathbf{z}), \dots, K_0(\mathbf{z}_{p_0}, \mathbf{z}))^\top$. Indeed, [26] shows that this vector preserves the Hilbertian inner-product in $\mathcal{H}_0$: $\langle \Pi_0 \varphi_0(\mathbf{z}), \Pi_0 \varphi_0(\mathbf{z}') \rangle_{\mathcal{H}_0} = \langle \psi_0(\mathbf{z}), \psi_0(\mathbf{z}') \rangle_{\mathbb{R}^{p_0}}$ for any $\mathbf{z}, \mathbf{z}'$ in $\mathbb{R}^{4\times l}$, where Π_0 denotes the orthogonal projection onto $\mathcal{E}_0$ (see also Appendix A). As the projection only depends on dot products – between the mapped patch and anchor points – it can be computed using the kernel, without explicitly mapping either point to $\mathcal{H}_0$. The approximation also bears strong similarities with a convolution network: as $K_0(\mathbf{z}, \mathbf{z}')$ is a function of the inner-product $\langle \mathbf{z}, \mathbf{z}' \rangle$, $\psi_0(\mathbf{z})$ is formed by a convolution (inner-product of each patch $\mathbf{z}$ with an anchor point $\mathbf{z}_i$) followed by a pointwise non-linearity (Gaussian kernel). In this sense, the anchor points of the approximation play the role of convolution filters. After pooling, an input sequence $\mathbf{x}$ is then also represented as a vector $\psi(\mathbf{x})$ in $\mathbb{R}^{p_0}$, corresponding to a point in $\mathcal{E}_0$. As a consequence,

the optimal prediction function f can also be parametrized by a scalar vector $\mathbf{w}$ in $\mathbb{R}^{p_0}$ such that $f(\mathbf{x}) = \langle \mathbf{w}, \psi(\mathbf{x}) \rangle$ and $\Omega(f) = \|\mathbf{w}\|^2$. Finally, jointly learning $\mathbf{w}$ and the anchor points $\mathbf{z}_i$ amounts to optimizing both the optimal subspaces and prediction function. This is achieved by stochastic optimization of (1) over $(\mathbf{z}_1, \dots, \mathbf{z}_{p_0})$, at the same computational cost per iteration as a classical convolutional neural network (see [26]). Specifically, we adopt an optimization scheme that alternates between two steps: (a) we fix the anchor points $(\mathbf{z}_i)_{i=1,\dots,p_0}$, compute the finite-dimensional representation $\psi(\mathbf{x}_1), \dots, \psi(\mathbf{x}_n)$ of all data points, and minimize (1) with respect to $\mathbf{w}$ using a convex optimization solver such as [12]; (b) We fix $\mathbf{w}$ and update all the $(\mathbf{z}_i)_{i=1,\dots,p_0}$ using one pass of a projected stochastic gradient descent (SGD) algorithm while fixing $\mathbf{w}$. The optimization for the reverse-complement formulation can be done in the same way except that it is no more convex with respect to $\mathbf{w}$, but we can still apply a fast optimization method such as L-BFGS [25], which is parameter-free. We find this alternating scheme to be more efficient than using an SGD algorithm jointly on $\mathbf{w}$ and the anchor points.

Multilayer construction. For simplicity, the previous construction was presented with a single layer, but the extension to multiple layers is straightforward. Instead of reducing the intermediate representation to a single point in $\mathcal{H}_0$, the pooling operation may simply reduce the sequence length by a constant factor, in a similar way as pooling reduces image resolution in [26]. Then, the previous process can be repeated and stacked several times, by defining a kernel $K_1, K_2, \dots$ on patches from the previous respective layer representations, along with Hilbert spaces $\mathcal{H}_1, \mathcal{H}_2, \dots$ and mapping $\varphi_1, \varphi_2, \dots$. When going up in the hierarchy, each point carries information from a larger sequence neighborhood with more invariance due to the effect of pooling layers [7].

2.3 Multitask Learning

In practice, several prediction tasks can sometimes be related, in which case sharing information across tasks can lead to better learning. For example, experiments can be conducted on the same TF by different laboratories, possibly across the same or different cell lines. The sequence data of the same TF may possess similar binding site patterns. Thus, it is natural to design a multitask learning scheme [8] that can take advantage of these common predictive patterns to improve the performance. Specifically, let us consider T tasks, which can represent T separate experiments involving the same TF. The multitask version of our CKN-seq has a single mapping function ψ across the different tasks, but task-specific predictors $(\mathbf{w}_t)_{t=1,\dots,T}$, as illustrated on Figure 1. The optimization formulation of this multitask problem can be written as

$$\min_{\mathbf{w}_1,\dots,\mathbf{w}_T \in \mathbb{R}^{p_k}, \mathbf{Z}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} L(y_{i,t}, \langle \mathbf{w}_t, \psi(\mathbf{x}_{i,t}) \rangle) + \frac{\lambda}{2} \|\mathbf{w}_t\|^2,$$

where $\mathbf{x}_{i,t}$ is a sequence from task t and $\mathbf{Z}$ represents the set of anchor points, which are shared across tasks. Learning is achieved by alternating minimization as in the single-task model.

2.4 Model Interpretation and Visualization

In this section, we propose a kernel-based method that can provide a PWM-like representation for each CKN-seq model, which allows to visualize models by generating corresponding sequence logos of PWMs. Without loss of generality, let us consider our single-layer model. The learned linear subspace $\mathcal{E}_0$ represents the Hilbert space in which the mapped sequences from distinct classes can best be linearly separated. Thus, the anchor points $\varphi_0(\mathbf{z}_i)$ can be viewed as the most representative points in $\mathcal{H}_0$ that best discriminate the sequences according to the labels. We propose two ways to

associate PWMs to our anchor points. The first one is to extract the sequence patches in a validation set of sequences which are sufficiently close to the anchor points in $\mathcal{E}_0$ and compute a PWM from these patches. This approach is similar to the one used in [1] – with an additional geometric interpretation, and requires a set of sequences to extract patches from. Another approach is to directly identify the closest PWM by projecting in the RKHS the image of the filter into the image of the set of PWMs. Using the kernel approximation, it amounts to solving $\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu}) - \psi_0(\mathbf{z}_i)\|^2$ for each $\mathbf{z}_i$, where $\mathcal{M}$ is the set of matrices in $\mathbb{R}^{4 \times l}$ with every column summing to 1. We solve this projection problem using a projected gradient descent algorithm. Thanks to the structure of simplex, this approach induces some sparsity to the resulting PWM and yields more informative logos. As opposed to the previous approach, it does not rely on a set of sequences.

3 Results

In this section, we empirically evaluate CKN-seq on a TF binding site prediction problem, exploiting the same ENCODE ChIP-seq data used in [1]. We start by quantitatively comparing our model to DeepBind in terms of area under the ROC curve (AUC) and speed, with the same network topology used in [1] for both methods. We then show how the performances of CKN-seq can be further improved by increasing the number of filters or using a multitask version of our model. We also show that a two-layer CKN-seq outperforms the one-layer CKN-seq, which is not the case for CNN [45]. Finally we show how the RKHS built by our trained model can be used to project the anchor points into the space of PWMs, yielding a simple interpretation in terms of motifs.

3.1 Datasets and Tasks

The problem of predicting TF binding sites is a classification problem, where the inputs are DNA sequences $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and the outputs are $y_1, \ldots, y_n$ in $\{+1, -1\}$ indicating whether or not the sequence binds a particular TF of interest. The positive sequences that we use come from the ENCODE ChIP-seq datasets[1] and are composed of the peaks in 506 different ENCODE ChIP-seq experiments from Uniform Peaks of Transcription Factor ChIP-seq. The negative sequences are generated by random dinucleotide shuffling as used in [1].

Following [1], we consider two ways of training models, differing by the number of training sequences used. The first one called *top* mode consists in training models on the top 500 odd-numbered peaks and then evaluating on the top 500 even-numbered peaks. The training can be done very fast in this mode as there is only small quantity of data which is supposed to contain the majority of binding site information. The other one called *all* mode consists in additionally using the remaining peaks to train models. As discussed in [1], using the remaining sequences typically leads to improved performance. Both modes are informative, shedding light on the behavior of the method with smaller or larger sample sizes respectively. For the sake of clarity and to save space, we restrict ourselves to the *all* mode experiments in this manuscript and move the *top* mode experiments to the appendix – except in the multitask learning section where we think discussing the effect of sample size is essential.

3.2 CKN-seq vs Convolutional Neural Network

We present the comparison between DeepBind and a CKN-seq with a logistic loss function L, in terms of prediction performance and runtime. As mentioned previously, DeepBind is a single-hidden-layer convolutional neural network. In order to make a fair comparison, we consider a

[1]Datasets are available at `http://tools.genes.toronto.edu/deepbind/nbtcode/`.

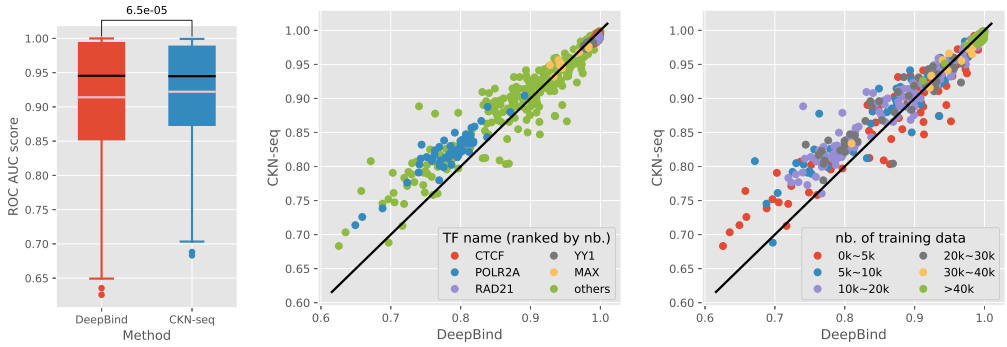


Figure 2: Comparison between DeepBind and CKN-seq in *all* mode. The pink and black line in the left figure respectively represent mean and median, the value on top is the p-value of a one-sided Wilcoxon signed-rank test. The TF names in the middle figure are ranked by the number of experiments in our whole datasets.

single-layer CKN-seq model with 16 filters as used for DeepBind in [1]. The filter size determines the length of the motifs learned by the network. As pointed out in [41], TF binding sites are typically 10 nt and we find indeed that models with filter size equal to 12 generally provide better performances. DeepBind uses longer filters of length 24, except for the RAD21, ZNF274, ZNH143 et SIX5 datasets where they use filters of length 32. In practice, DeepBind models learn longer patterns, part of which are the actual motif while the rest is noise. We will show later that our choice can lead to more informative sequence logos than DeepBind. In all our experiments, we use the Gaussian variant of the homogeneous dot-product kernel as detailed in the Appendix. As discussed in [29], the σ in our kernel can be interpreted as the tolerance of the mismatches in the patches and defines a neighborhood of mismatches in the RKHS $\mathcal{H}_0$. We fix this parameter to 0.3, according to a study of hyper-parameters described in Appendix C.2. Finally, the regularization parameter λ is fixed as 0.1 times the reciprocal of the number of training sequences.

We use the AUC to measure the prediction performance. To train CKN-seq, we split one quarter of the training data as validation data. We initialize the model using the unsupervised method presented in the appendix. We use the Adam algorithm [17] to update filters (fixing the weights of the logistic model) and L-BFGS algorithm[47] to optimize the logistic model, fixing the filter. The Adam optimizer starts with a learning rate equal to 0.1 and decrease by a factor of 2 when there is no decrease of validation loss since 4 epochs.

Prediction performance comparison. Figure 2 compares the AUCs obtained using our CKN-seq to the AUCs reported in [1] over the same datasets. The average ROC AUC score of CKN-seq on the total 506 experiments dominates the average score obtained by DeepBind. More precisely, CKN-seq generally outperforms DeepBind on the most difficult tasks, as illustrated on the right panel of Figure 2, where CKN-seq substantially improves the performance on the datasets where DeepBind shows poor performance.

Runtime comparison. As we observed that CKN-seq models are much less sensitive to hyper-parameters, they do not require an exhaustive calibrations step like DeepBind. In order to quantify the corresponding gain in speed, we measure the runtime of 30 experiments in *top* and *all* mode with CKN-seq and DeepBind on a Geforce GTX Titan Black, and report the results in Table 1. Training a CKN-seq model is about 25 times faster than a DeepBind model in *all* mode, and 70 times faster in *top* mode. Importantly, the DeepBind runtimes in Table 1 were obtained using the default 30 calibrations from the software package of [1], while the AUCs used in Figure 2 are the

	DeepBind *top*	CKN-seq *top*	DeepBind *all*	CKN-seq *all*
Runtime (min)	55.8 ± 0.5	0.8 ± 0.2	72.0 ± 1.0	2.9 ± 2.3

Table 1: Average runtime of training a model on one dataset for DeepBind and CKN-seq in *top* and *all* modes.

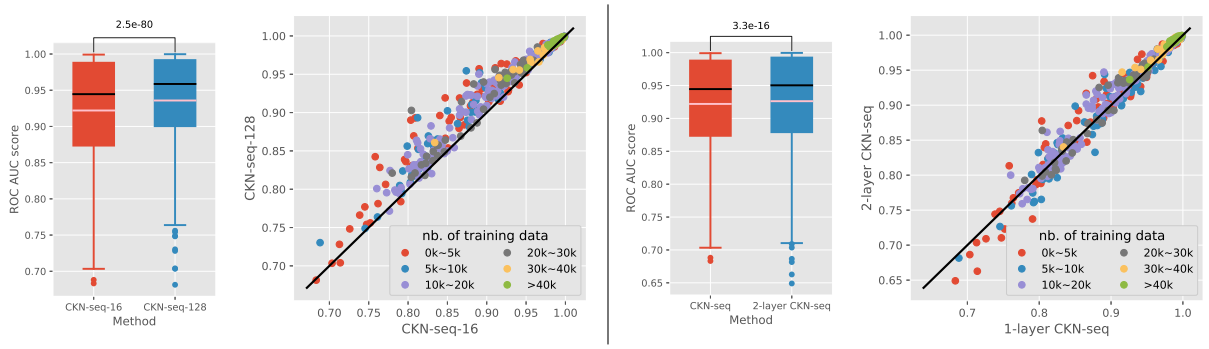


Figure 3: Comparison between CKN-seq with 16 filters and with 128 filters (left) and between one-layer and two-layer CKN-seq (right) for the *all* mode

ones reported in the same paper. In our experiments, obtaining these AUCs sometimes requires more than the default 30 calibrations, so that the average runtime would be even longer if one wanted to reproduce accurately the results displayed in Figure 2.

3.3 Influence of the Number of Filters

We increase the number of filters up to 128 and keep the other hyper-parameters unchanged except for the regularization parameter which we double. We present the results in the left panel of Figure 3, and observe that the CKN-seq-128 outperforms CKN-seq-16 on almost all the datasets, which is consistent with observations made for extensions of DeepBind in the literature [45]

We empirically observe that when parameterized with too many filters, some of them are learned by CKN-seq such that their feature representation in $\mathcal{H}_0$ is orthogonal to the image of ψ restricted to the set of motifs $\mathcal{M}$. We also observe that the representation of these patches have small coefficients at the positions corresponding to these anchor points and have therefore little impact on the final prediction. This is a possible explanation for the lack of overfitting incurred when increasing the number of filters. We further discuss this aspect in section 3.6 when we try searching the pre-images of some anchor points in $\mathcal{M}$.

3.4 Benefits of Multitask Learning

To assess the benefit of multitask learning, we group all the datasets of ENCODE ChIP-seq corresponding to the same TF and cell line, train our multitask CKN-seq model on these datasets and respectively compare the results to that obtained by a single-task model. We use the same hyper-parameters as used in the single-task model, and present the results in Figure 4 for *top* and *all* modes. In *top* mode, the multitask CKN-seq outperforms its single task counterpart on almost all the datasets, while the gain is more limited in *all* mode. This result is expected: sharing data across tasks helps significantly when the amount of data available for each task is limited – otherwise it may even add noise, as the shared data can be less specific.

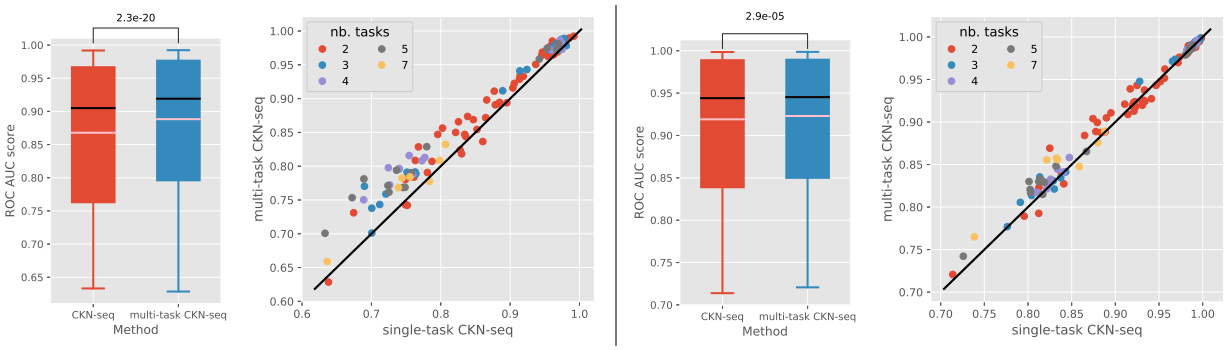


Figure 4: Comparison between single-task CKN-seq and multi-task CKN-seq for the *top* mode (left) and *all* mode (right). Note that the involved datasets are only those having common TF and cell line

3.5 Multilayer CKN-seq

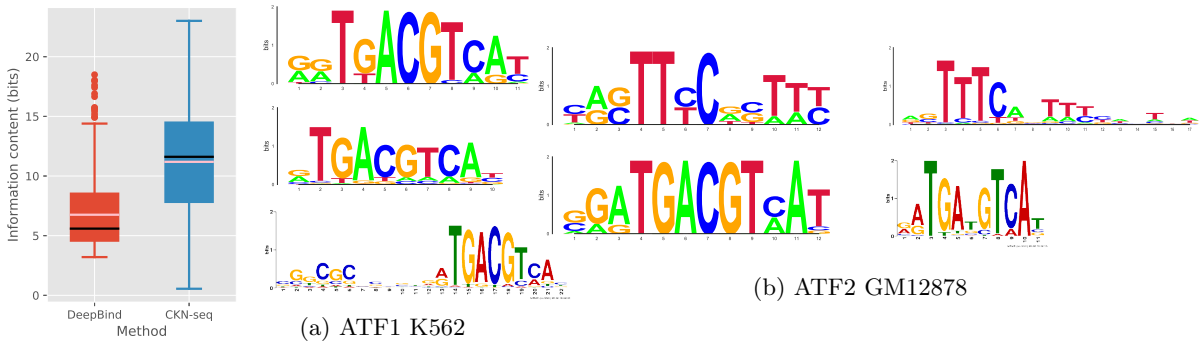
As discussed in [45], increasing the number of hidden layers in CNNs for biological sequences like DeepBind does not improve the prediction performances. By constrast, we show here that a two-layer CKN-seq may lead to higher AUCs when there is sufficient data. We use patch sizes of 12 and 3 for the first and second convolution layer respectively, number of filters equal to 64 and 16, and σ equal to 0.3 and 0.6. The results are presented in the right panel of Figure 3. As expected, the datasets on which the two-layer model performs worse than the one-layer counterpart generally have fewer training samples. With two layers, richer descriptors can be learned, which are however harder to interpret. When extending to even deeper models, CKN-seq may potentially capture co-motifs [38] although training deeper models is more difficult and requires large amounts of data. We intend to validate this on simulated data in the future.

3.6 Model Interpretation and Visualization

In this section, we study the ability of trained CKN-seq models to capture motifs and generate informative sequence logos. We consider here the single-layer CKN-seq models trained with the same hyper-parameters as presented in section 3.2, and use the second approach presented in section 2.4, *i.e.*, solving $\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu}) - \psi_0(\mathbf{z}_i)\|^2$ with a projected gradient descent method.

As discussed in 3.3, we observed that for some anchor points $\psi_0(\mathbf{z}_i)$ is orthogonal to the image of $\mathcal{M}$ by ψ_0. In this case $\arg\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu}) - \psi_0(\mathbf{z}_i)\|^2 = \arg\min_{\boldsymbol{\mu} \in \mathcal{M}} \|\psi_0(\boldsymbol{\mu})\|^2$ becomes a non-informative all 0.25 matrix regardless of $\mathbf{z}_i$. The corresponding filters obtained weights with small absolute value in the prediction function. Both observations provide us a way for removing non-informative filters, by (i) taking away the filters that don't have any elements of $\mathcal{M}$ in their mismatch neighborhood and (ii) filtering out anchor points with small weights in the prediction function (we use a threshold of $0.05\|\mathbf{w}\|_\infty$). We do not apply such a filter on the weights of DeepBind: since their motifs are built from validation sequences, they cannot be non-informative by construction.

We present the results in Figure 5. The average information content of DeepBind is obviously dominated by that of CKN-seq (left panel), although CKN-seq has larger variance as it doesn't use any sequence for validation and sometimes converges to non-existent motifs. Accordingly, the logos generated by CKN-seq are visually cleaner than the ones produced by the DeepBind representation, and we observe that they are also more informative than the ones proposed in the Factorbook database which was built on the same datasets [43]. This is illustrated on Figure 5a with the cAMP response element recovered by all three methods for ATF1, indeed known to bind to this element and validated in the curated Jaspar database [28]. This is also the case



(a) ATF1 K562

(b) ATF2 GM12878

Figure 5: Comparison of sequence logos between CKN-seq, DeepBind and Factorbook. Left: information content as measured in bits for the constructed PWMs. Middle: the same motif recovered by CKN-seq, DeepBind and Factorbook for ATF1; Right: motifs recovered CKN-seq (left), DeepBind (right top) and Factorbook (right bottom) for ATF2.

on Figure 5b with the same motif recovered for ATF2 by CKN-seq and Factorbook, along with another one recovered by CKN-seq and DeepBind. In addition as illustrated in the latter example, CKN-seq can sometimes capture motifs which were not recovered by DeepBind but were present in Factorbook. Like ATF1, ATF2 is known to bind to the cAMP response element. The motif recovered by CKN-seq and DeepBind but absent from Factorbook is not a known binder of ATF2, but is similar to other motifs described in Jaspar as binding other transcription factors in Human (SPI1, SPIB and NFATC2).

4 Discussion

We presented a CKN for biological sequences, and showed that it outperformed convolutional neural networks on a TF binding prediction task while being much faster to train. We also discussed how the CKN could be further improved by increasing the number of filters, the number of layers or by using a multitask scheme, and how its relying on a well defined function space could be used to produce a good interpretation of the trained model. We plan to validate more systematically the ability of our method to produce meaningful features by resorting to simulated sequences with known motifs and comparing to more methods [21, 36]. The fact that CKNs retain the ability of CNNs to learn feature spaces from large training sets of data – and actually can be interpreted as a particular form of CNNs – while enjoying an RKHS structure has other uncharted applications which we would like to explore in the future. First, it will allow us to leverage the existing literature on kernels for biological sequences for the first layer, capturing other aspects than sequence motifs. More generally, it makes it straightforward to build models for non-vector objects such as graphs, taking as input molecules or protein structures. Finally, it paves the way for making deep networks amenable to statistical analysis, in particular to hypothesis testing. This important step would be complementary to the interpretability aspect, and necessary to make deep networks a powerful tool for molecular biology beyond black box prediction.

Acknowledgements

This work was supported by grants from ANR (MACARON project ANR-14-CE23-0003-01 and FAST-BIG project ANR-17-CE23-0011-01) and by the ERC grant number 714381 (SOLARIS).

References

[1] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[2] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7):878, 2016.

[3] T. K. Attwood, M. E. Beck, D. R. Flower, P. Scordis, and J. Selley. The prints protein fingerprint database in its fifth year. *Nucleic Acids Research*, 26(1):304–308, 1998.

[4] T. L. Bailey. Dreme: motif discovery in transcription factor ChIP-seq data. *Bioinformatics*, 27(12):1653–1659, 2011.

[5] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences*, 91(3):1059–1063, 1994.

[6] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Computational Biology*, 4(10), 2008.

[7] A. Bietti and J. Mairal. Group invariance and stability to deformations of deep convolutional representations. *arXiv preprint arXiv:1706.03078*, 2017.

[8] R. Caruna. Multitask learning: A knowledge-based source of inductive bias. In *International Conference on Machine Learning (ICML)*, 1993.

[9] D. Castelvecchi. Can we open the black box of AI? *Nature*, 538(7623):20–23, 2016.

[10] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning (ICML)*. ACM, 2008.

[11] A. Drouin, S. Giguère, M. Déraspe, M. Marchand, M. Tyers, V. G. Loo, A.-M. Bourgault, F. Laviolette, and J. Corbeil. Predictive computational phenotyping and biomarker discovery using reference-free genome comparisons. *BMC Genomics*, 17(1):754, 2016.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 9(Aug):1871–1874, 2008.

[13] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 177–185, 1989.

[14] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology (JCB)*, 7(1-2):95–114, 2000.

[15] A. Jha, M. R. Gazzara, and Y. Barash. Integrative deep models for alternative splicing. *Bioinformatics*, 33(14):274–282, 2017.

[16] D. R. Kelley, J. Snoek, and J. L. Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, 26(7):990–999, 2016.

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] P. W. Koh, E. Pierson, and A. Kundaje. Denoising genome-wide histone chip-seq with convolutional neural networks. *Bioinformatics*, 33(14):i225–i233, 2017.

[19] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

[20] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 950–957, 1992.

[21] J. Lanchantin, R. Singh, B. Wang, and Y. Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. *arXiv preprint arXiv:1608.03644*, 2016.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[23] H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[24] C. Leslie, E. Eskin, J. Weston, and W. Noble. Mismatch String Kernels for SVM Protein Classification. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[25] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

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

[27] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2627–2635, 2014.

[28] A. Mathelier, O. Fornes, D. J. Arenillas, C.-y. Chen, G. Denay, J. Lee, W. Shi, C. Shyr, G. Tan, R. Worsley-Hunt, et al. Jaspar 2016: a major expansion and update of the open-access database of transcription factor binding profiles. *Nucleic Acids Research*, 44(D1):D110–D115, 2016.

[29] A. Morrow, V. Shankar, D. Petersohn, A. Joseph, B. Recht, and N. Yosef. Convolutional kitchen sinks for transcription factor binding site prediction. *arXiv preprint arXiv:1706.00125*, 2017.

[30] N. Murray and F. Perronnin. Generalized max pooling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[31] F. Picard, J.-C. Cadoret, B. Audit, A. Arneodo, A. Alberti, C. Battail, L. Duret, and M.-N. Prioleau. The spatiotemporal program of DNA replication is associated with specific combinations of chromatin marks in human cells. *PLoS Genetics*, 10(5):e1004282, 2014.

[32] H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

[33] B. Schölkopf, R. Herbrich, and A. Smola. A generalized representer theorem. In *Computational Learning Theory*. Springer, 2001.

[34] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[35] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[36] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning (ICML)*.

[37] A. Shrikumar, P. Greenside, and A. Kundaje. Reverse-complement parameter sharing improves deep learning models for genomics. *bioRxiv*, page 103663, 2017.

[38] S. Sinha and M. Tompa. Discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 30(24):5549–5560, 2002.

[39] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning (ICML)*, 2000.

[40] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[41] A. J. Stewart, S. Hannenhalli, and J. B. Plotkin. Why transcription factor binding sites are ten nucleotides long. *Genetics*, 192(3):973–985, 2012.

[42] G. D. Stormo. DNA binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

[43] J. Wang, J. Zhuang, S. Iyer, X.-Y. Lin, M. C. Greven, B.-H. Kim, J. Moore, B. G. Pierce, X. Dong, D. Virgil, et al. Factorbook. org: a wiki-based database for transcription factor-binding data generated by the encode consortium. *Nucleic Acids Research*, 41(D1):D171–D176, 2012.

[44] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 682–688, 2001.

[45] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford. Convolutional neural network architectures for predicting DNA–protein binding. *Bioinformatics*, 32(12):i121–i127, 2016.

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

[47] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

Appendices

A Construction Details of CKN-seq

In Sections A.1 and A.2, we present mathematical details underlying the convolutional kernel network model. In Section A.3, we discuss the interpretation of the σ parameter of the Gaussian kernel.

A.1 Choice of basic kernel K_0

We choose the homogeneous dot-product kernel introduced by [26] for all our experiments. Specifically, K_0 is defined, for any non-zero vectors $\mathbf{z}, \mathbf{z}'$ in $\mathbb{R}^{4\times l}$, by

$$K_0(\mathbf{z}, \mathbf{z}') = \|\mathbf{z}\|\|\mathbf{z}'\|\kappa_0\left(\frac{\langle \mathbf{z}, \mathbf{z}'\rangle}{\|\mathbf{z}\|\|\mathbf{z}'\|}\right), \quad \text{with } \kappa_0(1) = 1, \tag{2}$$

and $K_0(\mathbf{z}, \mathbf{z}') = 0$ otherwise. Here κ_0 is smooth and admits a Taylor expansion with non-negative coefficients to ensure the positive definiteness of K_0. In particular, we focus on the Gaussian kernel defined on the unit sphere: for all unit vectors $\mathbf{z}, \mathbf{z}'$

$$\kappa_0(\langle \mathbf{z}, \mathbf{z}'\rangle) := e^{\frac{1}{\sigma^2}(\langle \mathbf{z}, \mathbf{z}'\rangle - 1)} = e^{-\frac{1}{2\sigma^2}\|\mathbf{z}-\mathbf{z}'\|^2}.$$

A.2 Nyström Approximation for φ_0

Let us consider a patch $\mathbf{z} \in \mathbb{R}^{4\times l}$. The orthogonal projection $\varphi_0(\mathbf{z})$ onto the the finite-dimensional subspace $\mathcal{E}_0 := \text{span}(\varphi_0(\mathbf{z}_1), \ldots, \varphi_0(\mathbf{z}_{p_0}))$, where the $\mathbf{z}_i$ are anchor points with unit norms in $\mathbb{R}^{4\times l}$ called filters, is given by

$$\Pi_k\varphi_0(\mathbf{z}) = \sum_{i=1}^{p_0} \alpha_i \varphi_0(\mathbf{z}_i) \text{ with } \boldsymbol{\alpha} \in \underset{\boldsymbol{\beta}\in\mathbb{R}^{p_0}}{\arg\min} \left\| \sum_{i=1}^{p_0} \beta_i \varphi_0(\mathbf{z}_i) - \varphi_0(\mathbf{z}) \right\|^2, \tag{3}$$

which leads to solution

$$\boldsymbol{\alpha} = \mathbf{K}_{ZZ}^{-1}\mathbf{K}_Z(\mathbf{z}),$$

where $\mathbf{K}_{ZZ}^{-1}$ is the inverse (or pseudo inverse) of the $p_0 \times p_0$ Gram matrix $[K_0(\mathbf{z}_i, \mathbf{z}_j)]_{ij}$ and $\mathbf{K}_Z(\mathbf{z}) = (K_0(\mathbf{z}_1, \mathbf{z}), \ldots, K_k(\mathbf{z}_{p_0}, \mathbf{z}))^\top$. Then the projected vector $\Pi_k\varphi_0(\mathbf{z})$ admits a natural parametrization

$$\psi_0(\mathbf{z}) := \mathbf{K}_{ZZ}^{-1/2}\mathbf{K}_Z(\mathbf{z}) \in \mathbb{R}^{p_0},$$

in the sense that this p_0-dimensional vector preserves the Hilbertian inner-product in $\mathcal{H}_0$

$$\langle \Pi_k\varphi_0(\mathbf{z}), \Pi_k\varphi_0(\mathbf{z}')\rangle_{\mathcal{H}_0} = \sum_{i,j=1}^{p_0} \alpha_i\alpha'_j \langle \varphi_0(\mathbf{z}_i), \varphi_0(\mathbf{z}_j)\rangle_{\mathcal{H}_0} = \boldsymbol{\alpha}^\top \mathbf{K}_{ZZ}\boldsymbol{\alpha}' = \langle \psi_0(\mathbf{z}), \psi_0(\mathbf{z}')\rangle,$$

for any $\mathbf{z}, \mathbf{z}'$ in $\mathbb{R}^{4\times l}$.

In particular, the approximation for the homogeneous dot-product kernel is given by

$$\psi_0(\mathbf{z}) = \|\mathbf{z}\|\kappa_0(\mathbf{Z}_0^\top\mathbf{Z}_0)^{-1/2}\kappa_0\left(\frac{\mathbf{Z}_0^\top\mathbf{z}}{\|\mathbf{z}\|}\right) \text{ if } \mathbf{z} \neq 0 \text{ and } 0 \text{ otherwise}, \tag{4}$$

where $\mathbf{Z}_0 = (\mathbf{z}_1, \ldots, \mathbf{z}_{p_0})$ and κ_0 is, by abuse of notation, applied point-wise to its arguments.

A.3 Mismatch Neighborhood for Filters in the Case of Gaussian Kernel

Without loss of generality, let us consider the single-layer model and a point $\mathbf{z}$ in $\mathbb{R}^{4\times l}$. For unit-norm vectors, the Gaussian homogeneous dot-product kernel is the same as the Gaussian kernel, which we consider in this section. Our objective is to quantify a neighborhood around $\mathbf{z}$ such that the image of this neighborhood is

still a neighborhood of $\varphi_0(\mathbf{z})$ in $\mathcal{H}_0$. More precisely, let us consider a ball $\mathcal{B}$ centered at $\mathbf{z}$ with radius R and a random variable U uniformly distributed in $\mathcal{B}$. We now want to find a radius R such that the accumulated similarity in the neighborhood $\mathcal{B}$ dominates the whole similarity, *i.e.*,

$$\frac{\mathbb{E}[K_0(U,\mathbf{z})]}{\int K_0(\mathbf{x},\mathbf{z})dx} \geq \tau, \tag{5}$$

where τ is the percentile that we target. The left term can be simplified as

$$\frac{\mathbb{E}[K_0(U,\mathbf{z})]}{\int K_0(\mathbf{x},\mathbf{z})dx} = \int_{\mathbf{u}\in\mathcal{B}} q(\mathbf{x})dx = \mathbb{P}\left[\|X-\mathbf{z}\|^2/\sigma^2 < R^2\right],$$

with q the density function of a normal distribution $\mathcal{N}(\mathbf{z},\sigma^2\mathbf{I})$ and X is a random variable following this distribution. This term is the cumulative distribution function of a Chi-square distribution, which leads to

$$R^2 \leq F^{-1}(\tau, df(\mathbf{z})), \tag{6}$$

where F^{-1} is the inverse of cumulative distribution function of a Chi-square and *df* denote the degree of freedom.

Let us now return to our homogeneous Gaussian dot-product kernel and consider an anchor point $\mathbf{z}_i$ in $\mathbb{R}^{4\times l}$ which has unit norm. Observing that for any $\mathbf{x}\in\mathbb{R}^{4\times l}$

$$K_0(\mathbf{x},\mathbf{z}_i) = \|\mathbf{x}\|e^{\frac{1}{2\sigma^2}(\langle\mathbf{x},\mathbf{z}\rangle-1)} = \|\mathbf{x}\|e^{\frac{1}{2\sigma^2}(\langle(\mathbf{Z}_0^\top\mathbf{Z}_0)^{-\frac{1}{2}}\mathbf{Z}_0^\top\mathbf{x},(\mathbf{Z}_0^\top\mathbf{Z}_0)^{-\frac{1}{2}}\mathbf{Z}_0^\top\mathbf{z}_i\rangle-1)}$$

allows us to reduce the degree of freedom to the number of filters minus one since the norm of $\mathbf{z}_i$ is unitary. By consequence, $R \leq F^{-1}(\tau, df(\mathbf{z})) \leq F^{-1}(\tau, p_0 - 1)$ and the confidence radius is $F^{-1}(\tau, p_0 - 1)$.

B Optimization and Implementation Details

B.1 Choice of Hyper-Parameters

We detail here the choice of optimization hyper-parameters used in our experiments, including patch size l, σ in the homogeneous Gaussian dot-product kernel and regularization parameter λ.

Choice of patch size l. The patch size l determines the length of motifs learned by the network. In order to have an automatic choice without any prior information about the motif, one can carry out cross-validations or single validation via the loss on the validation set. Nevertheless, in our experiments we chose a fixed value $l = 12$ for all datasets in order to make a fair comparisons with DeepBind whose patch size is not cross-validated.

Choice of σ. This value determines the diameter of the mismatch neighborhood and therefore related to the tolerance to mismatches. We found the optimal value of σ to be quite consistent across different datasets. One can just fix this value or validate in *top* mode and then keep it fixed when using all the data (mode *all*). The value we choose in experiments is 0.3 at first layer and 0.6 for the higher layers.

Choice of regularization parameter λ. In practice, CKN-seq models are not very sensitive to this parameter above a certain threshold, as long as enough data is available. However, when data is scarce, the choice of this parameter is more subtle and may need to be selected by cross validation. In our experiments, we always kept to 0.1 times the reciprocal of the number of training data in order to make fair comparisons (the corresponding regularization parameter in deep learning model is called "weight decay", and is typically not cross-validated).

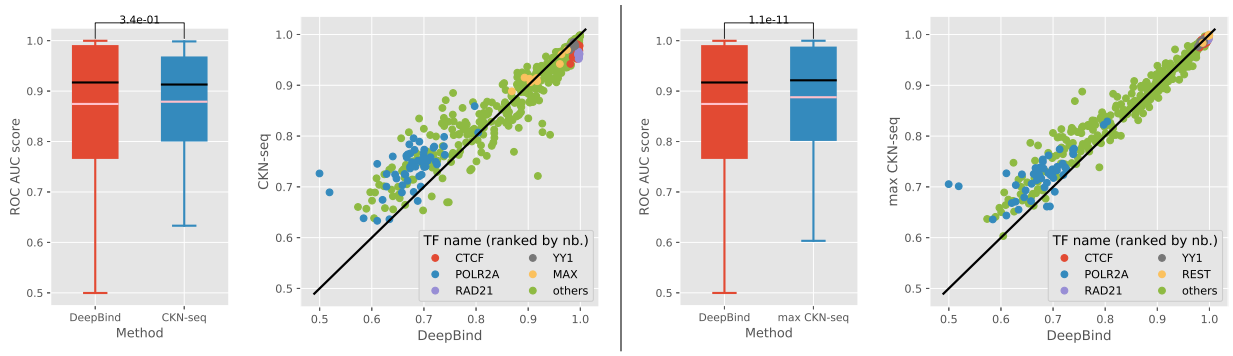


Figure 6: Comparison between DeepBind and CKN-seq for *top* mode: DeepBind vs CKN-seq with average pooling (left) and DeepBind vs CKN-seq with max pooling (right)

B.2 Initialization with the Unsupervised CKNs

Besides the *supervised* setting that we presented in section 2, CKNs can also be trained in an *unsupervised* way. In this case, the anchor points are obtained by running a K-means clustering algorithm on the set of patches, which can be randomly subsampled from the training sequences (see [26]). The number of filters corresponds to the number of clusters. This method usually provides similar performance to *supervised* learning method but with a very large number of filters, making interpretation more difficult. In practice, we use this way to initialize the filters for supervised learning. The main advantage of this initialization procedure is to be parameter-free. In contrast, convolutional neural networks are typically initialized with random weights, but require several hyper-parameters to be specified regarding the chosen random distribution.

C Additional Experiments

We now present results obtained on additional experiments.

C.1 Results for the Top Mode

Although the *top* mode performance is generally lower than the *all* mode performance, it is still interesting to understand the behavior of CKN-seq when few training samples are available. Analogous to the comparisons conducted for for the *all* mode, we present here the performance comparison between CKN-seq and DeepBind, the influence of the number of filters and the performance of multilayer CKN-seq. All the settings are kept the same as for mode *all*.

CKN-seq vs convolutional neural networks. We show the performance comparison between CKN-seq and DeepBind on the left panel of Figure 6. However, CKN-seq only performs slightly better than DeepBind in terms of mean AUC and even worse for median. As observed in the *all* mode, CKN-seq generally outperforms DeepBind on more difficult task, while DeepBind performs as well as CKN-seq on easy tasks, such as the datasets for CTCF. On the other hand, we found that a maximum (max) pooling as used in DeepBind, for aggregating patch representations into global representations, is helpful for the easier tasks in *top* mode. This is shown in the right panel of Figure 6. A typical interpretation for max pooling is that it can be viewed as the vector whose inner-product with each patch representation in $\mathcal{H}_0$ is constant, as described in [30] in the case of binary vectors. On ENCODE ChIP-seq datasets, we found max pooling usually gives better results in *top* mode while average pooling is better in *all* mode. A possible explanation is that when we do not have enough samples, we may learn filters corresponding to motifs but not precisely. Consequently, an average pooling scatters the contributions of patches to the motifs and thus confuses the classifier while a max pooling retains the maximal contribution of patches to the motifs. This is not true anymore when we have enough data to learn the filters corresponding to the exact motifs.

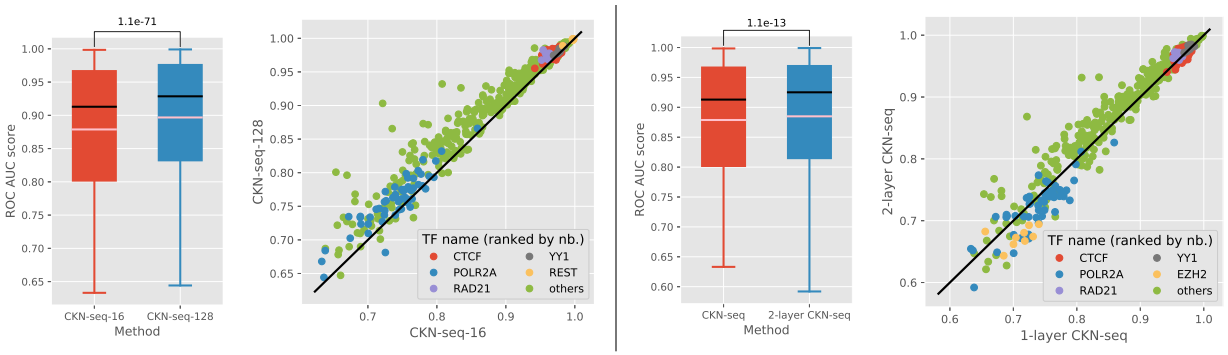


Figure 7: Comparison between CKN-seq with 16 filters and with 128 filters in *top* mode (left) and comparison between one-layer CKN-seq and two-layer CKN-seq in *top* mode (right)

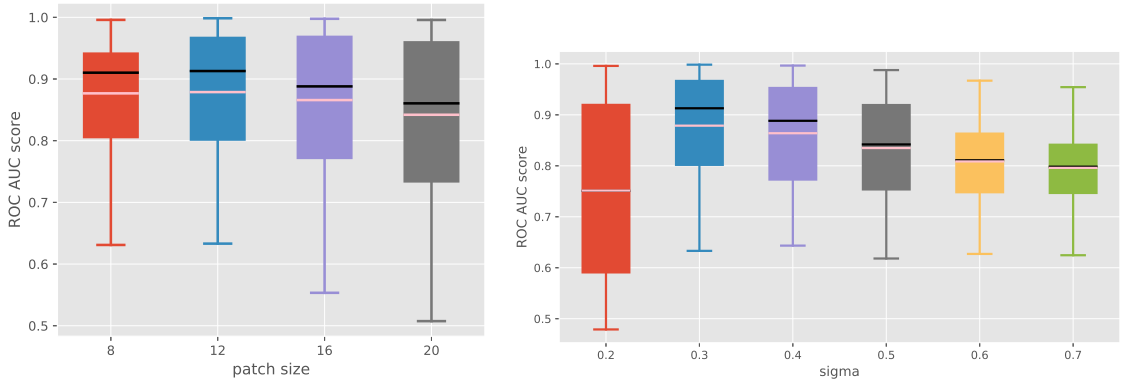


Figure 8: The impact of the choice of hyper-parameters on prediction performance in *top* mode: influence of patch size l while fixing $\sigma = 0.3$ (left) and influence of kernel parameter σ while fixing patch size $l = 12$ (right)

Influence of the number of filters. We obtain similar behaviors to *all* mode when increasing the number of filters, as shown in the left panel of Figure 7.

Multilayer CKN-seq. Similar results are obtained here in the right panel of Figure 7.

C.2 Study of Hyper-Parameters

As the behavior is similar in *all* mode, we show here the impact of the value of σ and patch size on prediction performance in *top* mode in Figure 8.