gkm-DNN: efficient prediction using gapped *k*-mer features and deep neural networks

Zhen Cao[1] and Shihua Zhang[1,2*]

[1]National Center for Mathematics and Interdisciplinary Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China; [2]School of Mathematics Sciences, University of Chinese Academy of Sciences, Beijing 100049, China.

*To whom correspondence should be addressed.

Abstract

How to extract informative features from genome sequence is a challenging issue. Gapped *k*-mers frequency vectors (gkm-fv) has been presented as a new type of features in the last few years. Coupled with support vector machine (gkm-SVM), gkm-fvs have been used to achieve an effective sequence-based prediction (e.g., transcription factor binding site prediction). However, the huge computation of a large kernel matrix prevents it from using large amount of data. To this end, we proposed a flexible and scalable framework gkm-DNN to achieve feature representation and prediction from high-dimensional gkm-fvs using deep neural networks (DNN). We first implemented an efficient method to calculate the gkm-fv of a given sequence. We then adopted a DNN model with gkm-fvs as input to achieve a prediction task. Here, we took the transcription factor binding site prediction as an illustrative application. We applied gkm-DNN onto 467 small and 69 big human ENCODE ChIP-seq datasets to demonstrate its performance and compared it with the state-of-the-art method gkm-SVM. We demonstrated that gkm-DNN can not only overcome the drawbacks of high dimensionality, colinearity and sparsity of gkm-fvs, but also make comparable overall performance and distinct better accuracy compared with gkm-SVM in much shorter time. Moreover, gkm-DNN can be easily adapted to other applications and combine different types of data using computational graphs.

Availability: All source codes of gkm-DNN are available at http://page.amss.ac.cn/shihua.zhang/.

Contact: zsh@amss.ac.cn.

1 Introduction

It is still a major challenge to study the function of primary DNA sequences in non-coding regions (Alexander *et al.*, 2010; Mercer *et al.*, 2009). The regulatory elements located in these regions usually contain several transcription factor binding sites (TFBSs), whose activities regulate multiple biological processes such as gene expression (Kasowski *et al.*, 2010). More importantly, some genetic variations in regulatory elements cause hereditary disease, which attracts more attention (Smyth *et al.*, 2008). In the past decade, several large-scale projects such as Encyclopedia of DNA Elements (ENCODE) and Roadmap Epigenomics Program have been launched to generate and profile large amounts of genome-wide data to figure out the mechanisms behind these regulatory processes (Bernstein *et al.*, 2010; Dunham *et al.* 2012).

In this background, using statistical and machine learning methods to study the regulatory elements is a promising paradigm. For example, Wang *et al.* (2012) used an expectation maximization (EM) based method to analyze position weight matrix (PWM) for 457 ENCODE ChIP-seq data sets of 119 different transcription factors. They profiled the sequence features and chromatin structure around the corresponding TFBSs. Liu *et al.* (2016) integrated oligomers of short length (known as *l*-mers) and six DNA local parameters to identify enhancers along with their intensity. Ghandi *et al.* (2014a) used gapped *k*-mer (oligomers of length *l* with $l - k$ non-informative positions) frequency vectors and support vector machines (gkm-SVM) to precisely predict the TFBSs. They also used the scores of gkm-SVM to successfully predict the impact of regulatory variants from DNA sequence (Lee *et al.*, 2015). More recently, Kelley *et al.* (2016) used one-hot coding format of DNA sequences and then applied deep convolutional neural networks (CNNs) to predict the chromatin accessibility of 164 cell types, and interpret some disease-related single nucleotide polymorphisms.

Yet, how to extract informative features from genome sequence is still a primary challenging issue despite the great success in studying the regulatory elements. The core of all above approaches is extracting features from genome sequences, while all of them have pros and cons. The position weight matrix (PWM) approach is close to the biological nature (e.g., motifs), but requires large amounts of data to determine appropriate scoring thresholds (Stormo, 2000). The one-hot coding format of DNA sequences resolves limited information explicitly, which puts the heavy pressure on model training (Alipanahi *et al.*, 2015; Kelley *et al.*, 2016; Zhou and Troyanskaya, 2015). As a supplement, using the frequencies of all *l*-mers to form feature vectors (*l*-mer frequency vector, abbreviated as *l*mer-fv) can resolve most of the information before training and prediction, while they becomes sparse and unstable as *l* increases (Ghandi *et al.*, 2014b). Compared to *l*mer-fvs, gapped *k*-mer frequency vectors (abbreviated as gkm-fvs) are not only close to the biological nature, but also more robust (Ghandi *et al.*, 2014b). Therefore, gkm-fv based methods have been used as benchmark methods for many bioinformatics problems (Kelley *et al.*, 2016; Qin and Feng, 2017; Zhou and Troyanskaya, 2015). However, there still exist limitations to be addressed when using gkm-fvs as features. For example, the dimensions of gkm-fvs grow exponentially as either *l* or *k* increases, which quickly become intractable. Besides, they are somehow sparse and highly collinear due to the inner structure of gkm-fvs (Ghandi *et al.*, 2014b). The popular method gkm-SVM adopts support vector machines with sophisticated kernel tricks to tackle the problems (Ghandi *et al.*, 2014a; Ghandi *et al.*, 2016). However, gkm-SVM cannot quickly deal with large amount of data due to inefficient computation of large-scale kernel matrix. More recently, LS-GKM (the large-scale gkm-SVM) has been invented to deal with large amount of data by skipping calculating the full kernel matrix (Lee, 2016). Yet, one cannot easily tune the hyper-parameters, since every time it is totally new to train another model. The shortcomings of kernel tricks may hinder the advantage of gkm-fvs. Thus, efficient methods are still needed to make full use of gkm-fvs.

In the last few years, the fast development of deep neural networks (DNNs) attracts great attentions in bioinformatics community due to several reasons. First, the rapid accumulation of diverse biological data fits in with the usability of DNN models. Second, the use of graphics processing unit (GPU) makes the training process of DNNs extremely faster than before (Ciregan *et al.*, 2012; Coates *et al.*, 2013). Third, it is easier to train models with the help of DNN specific technologies such as dropout and batch normalization (Hinton *et al.*, 2012; Ioffe and Szegedy, 2015). Therefore, DNNs have achieved state-of-the-art performance in a wide range of applications such as image classification and speech recognition (Hinton *et al.*, 2012; Krizhevsky *et al.*, 2012). In addition, bioinformatics community has also successfully applied DNNs to the predictions of transcription factor binding sites, genome variants, gene expression and so on (Alipanahi *et al.*, 2015; Chen, *et al.*, 2016; Zhou and Troyanskaya, 2015). The power of DNN in handling extremely large-scale datasets and learning hierarchical nonlinear data representation enables it has the potential to make full use of gkm-fvs.

To this end, we proposed a flexible and scalable method gkm-DNN to achieve feature representation and prediction from high-dimensional gkm sequence features using DNN (Fig. 1). We first implemented an efficient method to calculate the gkm-fvs (Fig. 1A). Next we took the gkm-fvs as input for DNN to achieve a prediction task (Fig. 1B). We took the TFBS prediction as an illustrative example, which is a widely studied problem. We demonstrated that gkm-DNN can not only overcome the drawbacks of high dimensionality, colinearity and sparsity of gkm-fvs, but also make comparable overall performance and distinct better accuracy compared with gkm-SVM in much shorter time. This is the first time to directly use gkm-fvs as features without using kernel tricks.

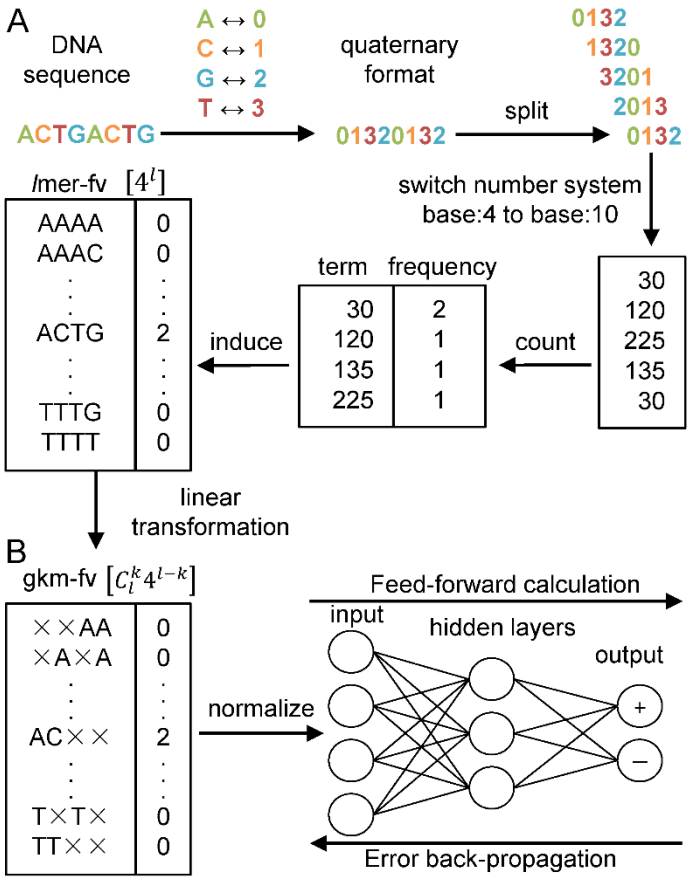

Fig. 1. The key components of constructing a gkm-DNN prediction model. (A) A quick method to calculate the gapped *k*-mer frequency vector (gkm-fv). (B) Illustration of a DNN model. After normalization, the gkm-fvs are taken as input for a multi-layer perceptron model. This

model is trained using the standard error back-propagation algorithm and mini-batch stochastic gradient descent method.

2 Materials and methods

2.1 Datasets

We downloaded all the 467 ENCODE ChIP-seq datasets used in (Ghandi *et al.*, 2014a) from http://www.beerlab.org/gkm-SVM/ to demonstrate the performance of gkm-DNN and compared it with gkm-SVM. All positive and negative sequences are the same as used in (Ghandi *et al.*, 2014a). Each dataset contains at most 5000 positive sequences and the same number of negative sequences.

We also collected 69 high-quality datasets of larger size for further tests from ENCODE (Dunham *et al.*, 2012). To ensure the quality of data, we downloaded all the datasets in the format of optimal idr thresholded peaks from the K562 cell lines. We only retained peaks with q-value < 0.01 and selected datasets with more than 25000 peaks. Given a dataset, we sort peaks first by -log(q-value), and then by its count in descending order. We only kept the top 20000 peaks for further analysis since there are always a lot of noises in ChIP-seq datasets (Park, 2009). If a peak is shorter than 301bp, we extended this peak from its midpoint to 301bp. If a peak is longer than 601bp, we cut down the peak to 601bp centered at the raw midpoint. After processing, we finally got 69 high-quality datasets. Each dataset contains 20000 positive samples.

Given positive samples, we also generated the same number of negative samples. We randomly sampled sequences of approximately the same length of positive samples from the whole genome. The randomly selected sequences were dropped if they have overlap with optimal idr thresholded peaks.

2.2 Calculating gkm-fvs

Oligomers of fixed length l are commonly known as l-mers. Gapped k-mers are l-mers with $l - k$ non-informative positions (Supplementary Fig. S1). In other words, for gapped k-mers, there are two parameters: word length l and matched position k ($k < l$, hence there are $l - k$ gaps). Given l and k, the frequencies of all gapped k-mers form a vector of length $C_l^k 4^{l-k}$, which is called gapped k-mer frequency vector (gkm-fv) in this paper. Different from the kernel computation of gkm-SVM, gkm-DNN directly uses the gkm-fvs as input. However, it is not a trivial task to calculate the gkm-fvs of hundreds of thousands of DNA sequences. Almost all methods use kernel tricks to skip its direct calculation. To the best of our knowledge, there is no quick program to directly calculate the gkm-fvs. For the first time, we implemented a fast method to calculate it.

The first step is to get l-mer frequency vector ($l$mer-fv), whose length is 4^l. Given a single strand DNA sequence s, we split it into length(s)-l+1 l-mers, and count their frequencies. We turn the l-mers into a quaternary representation (A→0, C→1, G→2, T→3), which induces their positions in the ordered $l$mer-fv to be a decimal integer (e.g., ACTGTCA→0132310 of base 4 →1972 of base 10). The computing using the quaternary representation is very fast, which skips the calculation of string matching. Hence, given a sequence, we can efficiently get its $l$mer-fv in a sparse manner.

The second step is to transform an $l$mer-fv to a gkm-fv with word length l. As illustrated in (Ghandi *et al.*, 2014b), an $l$mer-fv can fully determine a gkm-fv using a linear transformation. We calculate this transformation matrix and save it in a sparse manner. Next we use a sparse matrix multiplication to transform an $l$mer-fv to its corresponding gkm-fv.

In this paper, we consider the double strands DNA sequences. Instead of counting exact gapped k-mers, we also count their reverse complements and remove redundant gapped k-mers.

For example, two gapped *k*-mers ANNCTG and CAGNNT are reverse complements to each other and have exactly the same frequencies. Thus, we only kept one of them. This step not only reduces the input sizes, but also well captures the double strands information.

We normalize the frequency vectors before training the DNNs. Given a DNA sequence, we divide its gkm-fv by the length of the sequence. Then we multiply the frequencies with $\min(4^l / 2, 128)$ to reduce the floating point error caused by GPU calculation.

2.3 gkm-DNN model

Here we adopted a multi-layer feedforward neural network model to achieve the prediction task with the gkm-fv of a sequence as an input feature vector (Fig. 1B). It contains one input layer, one or multiple hidden layers and one output layer. All the layers are fully-connected. The feedforward calculation is a linear transformation with an activation function (Fig. 1B). Specifically, given the values of layer *m*-1, the values of layer *m* is calculated as:

$$O_m = f(W_{m-1}^T O_{m-1} + b_{m-1}), \qquad (1)$$

where O_{m-1} is the values of layer *m*-1, $\{W_{m-1}, b_{m-1}\}$ are the weights and the bias associated with layer *m*-1 that need to be learned. For the hidden layers, $f(\cdot)$ is the RELU activation function. The sigmoid activation function (or softmax function) is applied to the output unit for the classification purpose:

$$f(x) = \frac{1}{1+e^{-x}}. \qquad (2)$$

The cross entropy is used as the loss function for training, namely,

$$CE(y_i, \hat{y}_i) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)), \qquad (3)$$

where $\hat{y}_i$ is the output of an input sequence, $y_i = 1$ for a positive sequence, $y_i = 1$ for a negative sequence and N is the number of all training samples. Given the loss function and hyper-parameters (see below), the model is trained using the standard error back-propagation algorithm and mini-batch stochastic gradient descent method. We also used two tricks namely dropout and early stopping (Supplementary Materials and Supplementary Fig. S3). Passing all the training data through the model once is an epoch. Training is stopped when it hits the maximal training epoch given in advance.

2.4 Setup of gkm-DNN

There are some hyper-parameters for gkm-DNN. First, we set the word length l = 7 and the matched position k = 5 as default (see Results). The depth and width are different for datasets of different sizes. For the 467 small ENCODE ChIP-seq datasets (sample size: 10000), we only used one hidden layer and set the number of hidden nodes as $\{100, 400, 700\}$ for further analysis. For the 69 high-quality datasets (sample size: 40000), we set the number of hidden nodes as 700 and the number of hidden layers as $\{1, 2, 3\}$ for further analysis. For all models, we always applied dropout to the last two layers. The dropout rate was set as $\{10\%, 20\%, 30\%\}$ for further discussion. More details about hyper-parameters of gkm-DNN can be seen in the Supplementary Materials, Tables S1 and S2.

In this paper, we applied several practical ways to accelerate the training (Supplementary Materials). For input, we saved our data in a binary format in advance, which greatly reduces the time of input and output. For calculation, we used GPU (one NVIDIA GTX 1080) to train the model, which is about 10 times faster than using CPU alone (one Intel Xeon e5 1603 v3).

Given the training sequences, we first calculated the gkm-fvs (l = 7, k = 5) and then normalized them. Then we randomly split the total training set into a smaller training set and a validation set ($1/8$ of the total training set in this paper). For each group of hyper-parameters, we trained the gkm-DNN model using early stopping and got the best model for this group of hyper-parameters. Then we chose the best model for all groups of hyper-parameters according to their performances on the validation set. Note that once the total training set is given, our method will automatically give a best model. The validation set was only used internally in this method.

Given a test DNA sequence, we calculated the gkm-fv using corresponding l and k and normalized it. Next we used the best model to do prediction. The output is an estimated posterior probability on how likely this sequence is a positive one (i.e., how likely the transcription factor binds to this sequence). The default decision rule is set as: sequences whose output probabilities >= 0.5 are deemed as positive ones, otherwise negative ones.

2.5 Setup of gkm-SVM

In this paper, we used both the gkm-SVM R package and LS-GKM to implement gkm-SVM (Ghandi *et al.*, 2016; Lee, 2016). On both 467 small and 69 big ChIP-seq datasets, we trained gkm-SVM with l = 7, k = 5 and l = 10, k = 6 for comparison. Other parameters are set as defaults. We applied several practical ways to accelerate the training of gkm-SVM. For gkm-SVM R package, we first calculated the full kernel matrix and then used the cross-validation mode. This step greatly reduces the repetitive computation of a kernel matrix. In addition, the gkm-SVM tasks were also parallelized using R language. For LS-GKM, we used all the four cores (four threads) and 10G cache memory, which makes full use of our machine.

2.6 Performance assessment

We adopted multiple standard measures to comprehensively evaluate gkm-DNN and gkm-SVM. The receiver operating curves (ROC) and precision-recall curves (PRC) are two typical graphical plots that illustrate the classification ability of a binary classifier system as its discrimination threshold is varied (Davis and Goadrich, 2006; Fawcett, 2004). We used the area under the ROC and PRC curves (AUC and AUPRC). However, many studies have illustrated the drawbacks of ROC and AUC (Drummond and Holte, 2004; Lobo *et al.*, 2008). The areas under the curves essentially only assess the ability of ranking samples. One rarely uses AUC in a practical classifier because only a couple of points on the curves are useful (Lobo *et al.*, 2008). For a typical prediction task such TFBS prediction, it is very important to give a solid binary prediction (Bhardwaj *et al.*, 2005). Thus, we also adopted four widely used performance measures including

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}, \quad (4)$$

$$Precision = \frac{TP}{TP + FP}, \quad (5)$$

$$Recall = \frac{TP}{TP + FN}, \quad (6)$$

$$F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}, \quad (7)$$

where *TP*, *FP*, *TN* and *FN* represent true positives, false positives, true negatives and false negatives, respectively. In addition, we used the standard 5-fold cross-validation procedure to get the mean of above measurements.

2.7 Training DNN with different size of samples

We also trained gkm-DNN models with different size of training samples using the 69 high-quality datasets. For each dataset and each cross-validation fold, given the same 4000 validating samples, we used training samples of equal size (1X), twice (2X), four times (4X) and seven times (7X) to train the gkm-DNN models and calculated the AUCs and accuracies on the test sets. To strictly control the quality of data, 1X samples are contained in 2X samples, 2X samples are contained in the 4X samples and so on. Note that using 7X samples is the same as using all training samples as above. The maximum training epochs were set as 210, 105, 53 and 30 for 1X, 2X, 4X and 7X training samples. All other hyper-parameters were set as the same one.

2.8 Implementation

We used R language to calculate the gkm-fvs of DNA sequences and DL4J (deep learning for Java) framework to train neural networks. All the source codes are available at http://page.amss.ac.cn/shihua.zhang/.

3 Result

3.1 Evaluation of different gkm-fvs

The first problem for running gkm-DNN is to select proper *l* and *k* to determine gkm-fvs. Here, we used six high-quality datasets with different putative motif lengths according to JASPAR core (Sandelin *et al.*, 2004) to evaluate the effects of gkm-fvs in terms of *l* and *k* with dimensions less than 20000 due to computing burden (Supplementary Table S1).

Generally, different gkm-fvs in terms of *l* and *k* significantly influence the performance of gkm-DNN. Given *l*, the larger the *k* is, the better the performance is (Fig. 2). When *k* = 4 or 5, the performances are significantly better than that of *k* = 2 or 3 (Fig. 2). Thus, very small *k* cannot resolve enough information from DNA sequences. On the other hand, given *k*, the longer the *l* is, the higher the AUC is (Fig. 2). The results in terms of accuracy and F1-score show similar performance (Supplementary Fig. S3). Thus, we note that large *l* and proper *k* may resolve more information from raw DNA sequences directly. We can see that the results of *l* = 7, *k* = 5 (10752 features) and that of *l* = 9, *l* = 4 (16176 features) are comparable. Due to the computational efficiency, we recommend *l* = 7, *k* = 5 to run gkm-DNN as the defaults.

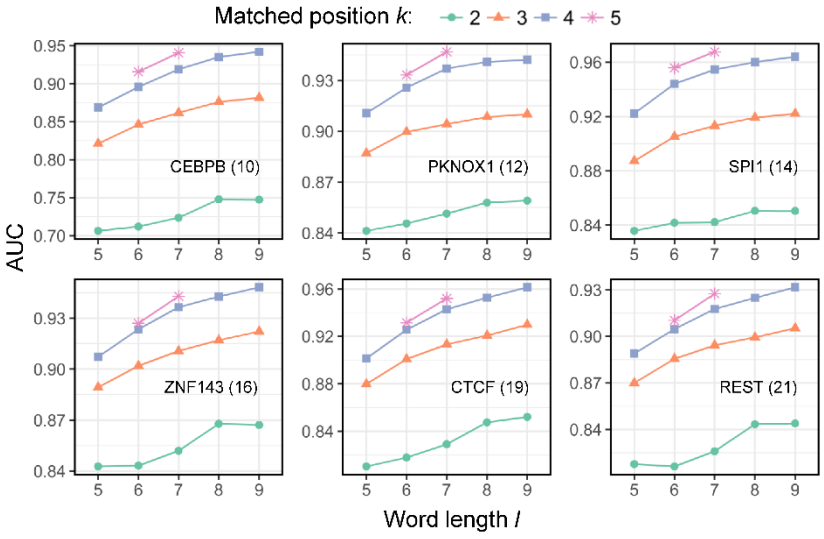

Fig. 2. Performance of different gkm-fvs on six representative datasets in terms of AUC (*y* -coordinate) with respect to different word length *l* (*x*-coordinate) and matched position *k*. Each point represents the AUC of a type of gkm-fvs. Lengths of putative motifs are shown in the parentheses. Gkm-fvs with dimensions larger than 20000 are ignored due to computing burden.

3.2 Performance on human ENCODE ChIP-seq datasets

Here we applied gkm-DNN onto the 467 human ENCODE ChIP-seq datasets and compared it with gkm-SVM 2.0. Combination of gkm-fvs and DNN demonstrate strong potentials. At first, we used the same gkm-fvs (l = 7, k = 5) for both methods (Fig. 3A). The two methods show very competitive performances in terms of AUCs and AUPRCs (Fig. 3A and Supplementary Fig. S4A). Surprisingly, gkm-DNN outperforms gkm-SVM on almost all datasets (466/467) in terms of accuracy and F1-score (Fig. 3A). Besides, gkm-DNN always has higher recalls and lower precisions on all the datasets (Supplementary Fig. S4A), suggesting that gkm-DNN prefers to classify samples as positive ones compared to gkm-SVM. Taken precisions and recalls together, gkm-DNN has higher F1-scores on all datasets, implying that the results of gkm-DNN are more balanced. We note that gkm-DNN essentially uses less training samples due to the internal validation.

We also compared gkm-DNN (l = 7, k = 5) with gkm-SVM using its default parameters (l = 10, k = 6). Currently, it is hard for gkm-DNN to use large amount of features due to computing burden (>20000). gkm-SVM indeed outperform gkm-DNN in terms of AUCs and AUPRCs in many cases (Fig. 3B and Supplementary Fig. S3). However, gkm-DNN still outperforms gkm-SVM on almost all the datasets in terms of accuracy and F1-score (Fig. 3B).

In general, gkm-DNN is more robust than gkm-SVM. On 386 datasets, gkm-DNN has lower AUCs, but higher accuracies and F1-scores than gkm-SVM. Thus, although the default cutoff 0 is the theoretically optimum value for gkm-SVM, it is not the optimal one for practical applications. The reason is that the best separation hyperplane trained from SVM also includes an intercept term. We can also observe this phenomenon for gkm-SVM with different gkm-fvs (Supplementary Fig. S4C).

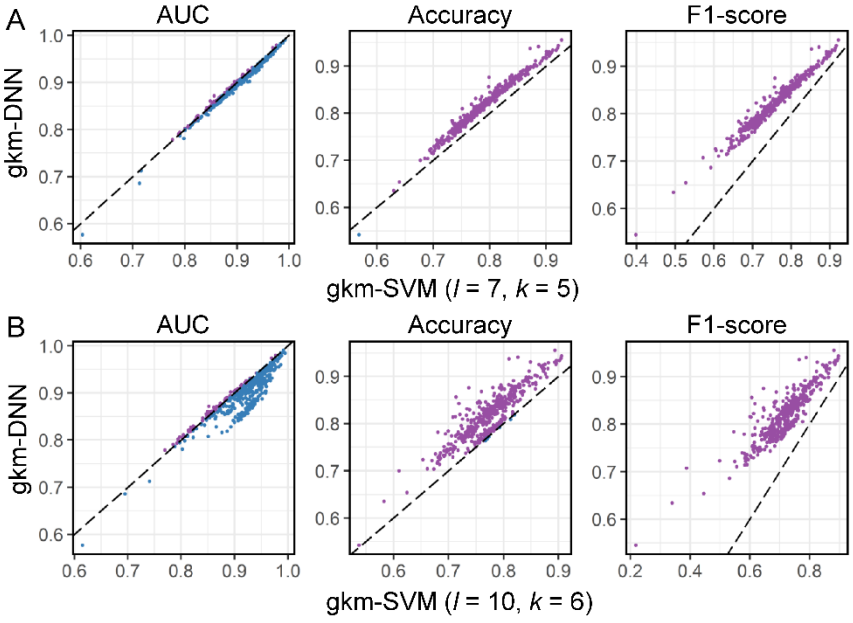

Fig. 3. Performance comparison between gkm-DNN and gkm-SVM on 467 small ChIP-seq datasets in terms of AUC, accuracy and F1-score. Each point represents the prediction results of gkm-DNN (*y*-coordinate) and gkm-SVM (*x*-coordinate) of a dataset. Points above and below the line $y = x$ are in purple and blue respectively. (A) Comparison using the same gkm-fvs (*l*

= 7, k = 5 for both gkm-DNN and gkm-SVM). (B) Comparison using the default gkm-fvs respectively (l = 7, k = 5 for gkm-DNN and l = 10, k = 6 for gkm-SVM).

3.3 Performance on high-quality datasets of larger size

Here we applied gkm-DNN onto 69 high-quality datasets with relative large sizes (40000 samples per dataset) and compared it with gkm-SVM. The results on these big datasets of gkm-DNN and gkm-SVM show very similar performance to those on the small datasets. When using the same features (l = 7, k = 5), the two methods are comparable in terms of AUCs and AUPRCs; while gkm-DNN are superior to gkm-SVM in terms of accuracy and F1-score (Fig. 4A). When gkm-SVM uses longer gapped k-mers (l = 10, k = 6), it can achieve higher AUCs and AUPRCs than gkm-DNN in many cases. However, gkm-DNN still gets higher accuracies and F1-scores in the most cases (Fig. 4B). In addition, gkm-DNN always has higher recalls and lower precisions than gkm-SVM.

Thus, gkm-DNN is more discriminative compared to gkm-SVM for a classification task. We recommend gkm-DNN if one needs an explicit classification, which is a fundamental goal of many prediction tasks. If the task is only care about ranking or further analysis does not relate to decision cutoff value, then gkm-SVM with longer gapped k-mers may be a better choice.

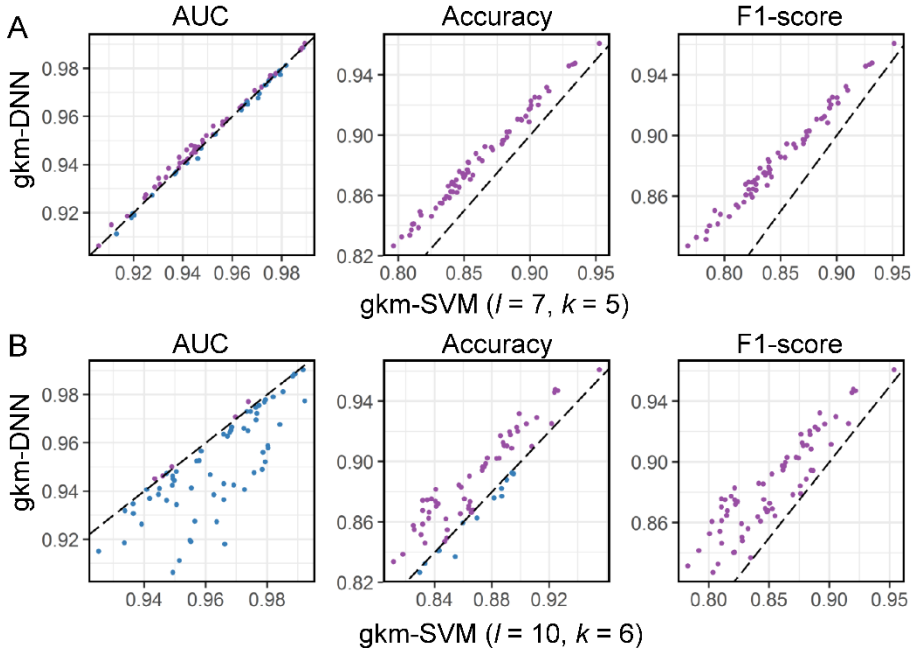

Fig. 4. Performance comparison between gkm-DNN and gkm-SVM on 69 high-quality datasets of larger sample sizes in terms of AUC, accuracy and F1-score. Each point represents the prediction results of gkm-DNN (y-coordinate) and gkm-SVM (x-coordinate) of a dataset. Points above and below the line $y = x$ are in purple and blue respectively. (A) Comparison using the same gkm-fvs (l = 7, k = 5 for both gkm-DNN and gkm-SVM). (B) Comparison using the default gkm-fvs respectively (l = 7, k = 5 for gkm-DNN and l = 10, k = 6 for gkm-SVM).

3.4 Computational efficiency

Besides prediction accuracy, we aimed to develop a fast method which can handle large-scale datasets. We tested the average runtime of gkm-DNN and gkm-SVM under default parameters using our desktop computer (Supplementary Table S4). On small datasets (sample size 10000), gkm-DNN is twice faster than gkm-SVM 2.0 and ten times faster than LS-GKM (Fig. 5A). The reason is that LS-GKM does not pre-save the kernel matrix and does many repetitive computation. On big datasets (sample size 40000), the gkm-SVM 2.0 package cannot handle

this case because it needs too much memory. Moreover, gkm-DNN is also ten times faster than LS-GKM (Fig. 5A).

More importantly, the runtime of gkm-DNN is proportional to the sample size. We explored the runtime of its three components, namely calculating gkm-fvs, saving gkm-fvs into binary format and training DNN (Fig. 5B). All the three parts can be naturally paralleled at the sample level. Thus, the runtime of each part should be proportional to sample size. We confirmed this by comparing the average runtime of gkm-DNN on small and big datasets (Fig. 5B). On big datasets, the training of DNN part is faster than expected because it can automatically take up more loading of GPU (37% on small datasets and 65% on big datasets). Once the loading is fixed, the runtime of this part is also proportional to sample sizes.

The efficient training of gkm-DNN is affected by several aspects. First, we implemented a quick method to calculate the gkm-fvs. Second, as to the training of DNN, once hyper-parameters are given, the training complexity is proportional to $O = E \times N \times G$, where E is the training epoch, N is the number of training samples and G is the length of gkm-fvs $(E = 30, N = 10752)$. Therefore, the training time of gkm-DNN is approximately proportional to the sample size, which is supported by our test (Fig. 5B). Third, we built the DNN blocks using DL4J, which has effective backend supporting CUDA 8.0. Fourth, we used a GPU (GTX 1080) to do most of the calculation, which is about 10 times faster than using our CPU (Intel Xeon e5 1603 v3). In the future, we believe that gkm-DNN will further benefit from the rapid development of both software (e.g., CUDA and MKL) and professional computing devices (e.g. NVIDIA GPU and Intel Xeon phi).

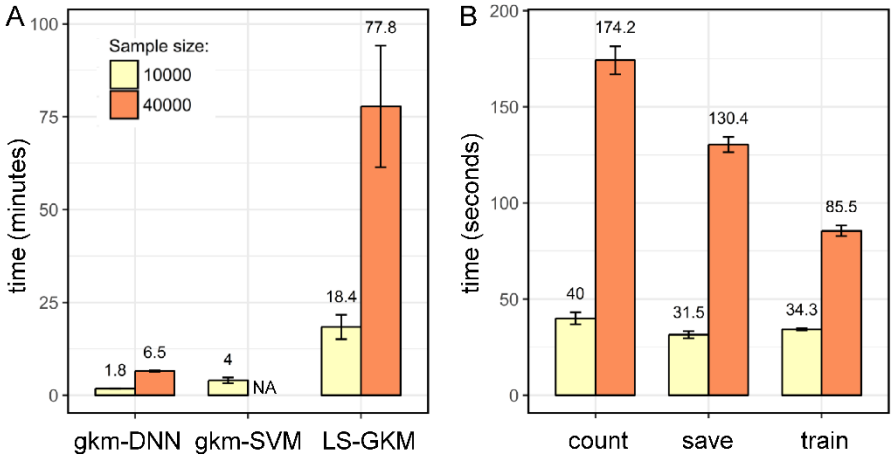

Fig. 5. Computational efficiency of gkm-DNN. (A) Comparison of computing time between gkm-DNN and gkm-SVM. The average runtime of both methods for one dataset using a desktop computer was presented. (B) Average runtime of key components of training gkm-DNN including calculating gkm-fvs (count), saving gkm-fvs (save) into binary format and training DNN (train) on small and big datasets respectively.

3.5 gkm-DNN has great superiority with more data

We observed that the performance on big datasets are greatly better than that on small datasets using either gkm-DNN or gkm-SVM (e.g., the average AUCs are 0.952 and 0.893 for gkm-DNN respectively) (Fig. 3 and 4). Therefore, we believe that larger sample size is also important in real applications. In this situation, gkm-DNN have great superiority because it can quickly make full use of large amount of data as shown above.

We trained gkm-DNN based on the 69 high-quality datasets using different number of training samples (Materials and Methods). We first observed that models trained from more data have overall higher AUCs and accuracy than models trained from less data (Fig. 6). Furthermore, for each dataset, the results of gkm-DNN are always better when using more data

(Supplementary Fig. S6). In short, gkm-DNN can achieve an improvement using more training samples. With rapid development of biological technology, more DNA sequence data will emerge and gkm-DNN will definitely benefit from these large-scale data.

3.6 Interpretation of gkm-DNN

We have demonstrated that gkm-DNN runs faster and makes very competitive prediction performance compared to gkm-SVM using on 536 datasets. Here, we attempted to elucidate the learned neural networks and figure out how gkm-DNN works using the 69 high quality datasets. For inputs, we can view the gkm-fvs as a type of auto-encoder of *l*mer-fvs (Supplementary Fig. S7). In practice, gkm-SVM uses gkm-fvs or estimated *l*mer-fvs based on gkm-fvs as input (Supplementary Fig. S7). We can view the input of gkm-DNN as a denoising version of *l*mer-fvs, which makes the classifier easy to learn.

Moreover, using hidden layers can overcome the drawbacks of high dimensionality, colinearity and sparsity of gkm-fvs. Compared to 10752 raw features, the hidden layers only have 700 nodes (Supplementary Fig. S8A), which forces the models to learn the most representative features. Furthermore, the learned activation values of hidden nodes are not redundant. Given a trained model and all training samples, we calculated the condition numbers of the activation values of the last hidden layer. The condition numbers are all very small (Supplementary Fig. S8B), indicating that the features extracted by hidden layers are not collinear at all. In addition, the activation values of the last hidden layer are denser than the gkm-fvs. Although these values contain zeros due to the RELU activation function, the ratio of zeros is distinctly lower than that of gkm-fvs (Supplementary Fig. S8C).

The hidden layers learn useful patterns for the classification task. Here we took one learned model as an example. For the last hidden layer, we clustered both the 700 hidden nodes and the 32000 training samples using the activation values (Supplementary Fig. S8D). There are clear clusters in the heatmap, which distinguish a fraction of training samples from others (Supplementary Fig. S8D). We also employed t-distributed stochastic neighbor embedding (t-SNE) to transform the 700 features into two representative directions (dimensions). The two directions can separate many positive samples from negative samples distinctly (Supplementary Fig. S8E). Moreover, the distribution of negative samples are more disperse than that of positive ones. The negative samples are randomly selected from the whole genome, implying that they may contain no much strong signals. In contrast, the positive samples may contain some underlying patterns (e.g., motifs). Thus, gkm-DNN distinguishes positive samples using the combination of some weak patterns from the negative ones.

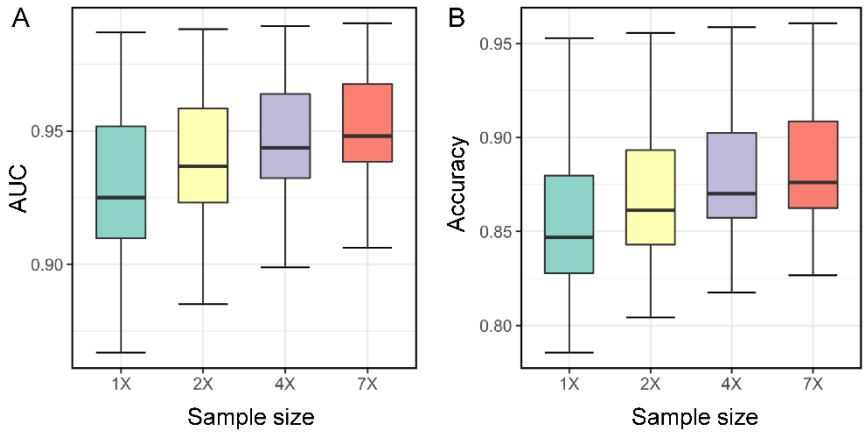

Fig. 6. Performance comparison of gkm-DNN trained using different sample sizes in terms of AUC. For each dataset and each cross-validation fold, given the same validation set, training sets of equal size (1X), twice (2X), four times (4X) and seven times (7X) were used to train

gkm-DNN. For each dataset, average AUCs (A) and accuracies (B) were calculated on the test sets for five-fold cross-validation.

4 Discussion

In this paper, we presented a flexible and scalable method gkm-DNN to achieve a sequence-based prediction task. We first implemented a quick method to calculate the gkm-fvs, and then used them as input to train a DNN model. We took the widely studied TFBS prediction problem as an illustrative example. We evaluated gkm-DNN and compared it with the state-of-the-art gkm-SVM using 467 small and 69 big datasets. We showed that gkm-DNN can not only overcome the high dimensionality, colinearity and sparsity of gkm-fvs, but also make competitive performance compared to gkm-SVM in much shorter time.

It is a fundamental problem to give a solid and deterministic prediction whether the transcription factor binds to a given sequence or not. For traditional PWM based methods, it is very hard to determine a proper cutoff value (Stormo, 2000). Although gkm-DNN and gkm-SVM have the abilities to automatically give good cutoff values, they behave quite differently. To figure out the reasons, we calculated the best cutoff values to optimize accuracies on the test sets (note that you will never know these values in real applications). Default cutoff values of both methods are both approximated but not optimal in the most cases. However, default value 0 of gkm-SVM is higher than the best ones in more than half of the cases, while default value 0.5 of gkm-DNN is lower than the best ones in more than half of the cases (Supplementary Fig. S9A). This explains why gkm-DNN always has higher recalls but lower precisions than gkm-SVM. The higher F1-scores implies that the results of gkm-DNN are more balanced. Furthermore, gkm-DNN is more robust with the default cutoff value. We observed that the prediction values of gkm-DNN were almost concentrated at 0 and 1 while the prediction values of gkm-SVM were more disperse (Supplementary Fig. S9B). Hence, for gkm-DNN, the samples with a prediction value near 0.5 are very few, meaning that a small perturbation of cutoff value 0.5 will not influence most of the predictions. As to the higher AUC but lower accuracy phenomenon of gkm-SVM, we guess the simplified calculation of kernel matrix may result in the loss of information (Ghandi *et al.*, 2014a).

gkm-SVM and gkm-DNN are complementary to each other. gkm-SVM can use relative longer gapped *k*-mers by kernel tricks while it cannot handle large amount of samples. Although LS-GKM improves this (Lee, 2016), the running is still not efficient enough. On the other hand, gkm-DNN can deal with large amount of data using relatively shorter gapped *k*-mers. How to combine the advantages of both methods may be a valuable direction in the future. For gkm-SVM, it needs fast methods to calculate the inner product of two gkm-fvs. For gkm-DNN, it needs some practical skills to make full use of longer gapped *k*-mers.

One possible difficulty to use gkm-DNN is that one need to choose many hyper-parameters to train a model. The biggest factor affecting the performance is l and k for gkm-fvs. We recommend $l = 7$, $k = 5$ for typical cases. The hyper-parameters of training neural networks have diverse effects on different datasets (Supplementary Fig. S10A). However, if we only use the defaults rather than nine sets of hyper-parameters, the reductions of performance are very limited (Supplementary Figs. S10B and S10C). In conclusion, it is acceptable to use the well-chosen default hyper-parameters for gkm-DNN.

Validation sets are very useful for gkm-DNN. It is hard to determine whether the model is trained properly without under- nor over-fitting if the training epoch is given manually. Validation set are used to determine the best hyper-parameters and the real training epoch according to cross entropy. However, this step is at the cost of using less data during training. Several practical ways can improve this drawback. For example, using bagging method to

combine the models, which can use out-of-bag samples as validation set (Breiman, 1996). There may be no test sets for real applications, and we strongly recommend to use validation sets.

It is easy to extend gkm-DNN due to its high flexibility and scalability. First, one can uses large amount of data without waiting too long. Second, gkm-DNN can also add other data sources as inputs using a computational graph, which is a directed acyclic graph representing the information flow (Supplementary Fig. S11). The hidden layers automatically extract the high-level features and allocate different weights for different data sources. Third, gkm-DNN can deal with a variety of outputs such as multi-label classification and regression (Supplementary Fig. S11). The only change is to use a proper activation function for the output layer and a proper loss function for training. Therefore, we can easily adapt gkm-DNN to many other prediction tasks such as transcription factor binding affinity prediction, RNA binding protein prediction, enhancer recognition and gene expression prediction (Liu *et al.*, 2016; Pelossof *et al.*, 2015).

References

Alexander, R.P., *et al.* (2010) Annotating non-coding regions of the genome. *Nat. Rev. Genet.*, **11**(8), 559-571.

Alipanahi, B., *et al.* (2015) Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat. Biotechnol.*, **33**(8), 831-838.

Bernstein, B.E., *et al.* (2010) The NIH Roadmap Epigenomics Mapping Consortium. *Nat. Biotechnol.*, **28**(10), 1045-1048.

Bhardwaj, N., *et al.* (2005) Kernel-based machine learning protocol for predicting DNA-binding proteins. *Nucleic Acids Res.*, **33**(20), 6486-6493.

Breiman, L. (1996) Bagging predictors. *Mach. Learn.*, **24**(2), 123-140.

Chen, Y., *et al.* (2016) Gene expression inference with deep learning. *Bioinformatics*, **32**(12), 1832-1839.

Ciregan, D., *et al.* (2012) Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3642-3649.

Coates, A., *et al.* (2013) Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 1337-1345.

Dunham,I. *et al.* (2012) An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489, 57–74.

Davis, J. and Goadrich, M. (2006) The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233-240.

Drummond, C. and Holte, R.C. (2004) What ROC Curves Can't Do (and Cost Curves Can). In *ROCAI*, pp. 19-26.

Fawcett, T. (2004) ROC graphs: Notes and practical considerations for researchers. *Mach. Learn.*, **31**(1), 1-38.

Ghandi, M., *et al.* (2014a) Enhanced regulatory sequence prediction using gapped k-mer features. *PLoS Comput. Biol.*, **10**(7), e1003711.

Ghandi, M., *et al.* (2014b) Robust k-mer frequency estimation using gapped k-mers. *J. Math. Biol.*, **69**(2), 469-500.

Ghandi, M., *et al.* (2016) gkmSVM: an R package for gapped-kmer SVM. *Bioinformatics*, **32**(14), 2205-2207.

Hinton, G., *et al.* (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Proc. Mag.*, **29**(6), 82-97.

Hinton, G.E., *et al.* (2012) Improving neural networks by preventing co-adaptation of feature detectors. *arXiv Preprint arXiv:1207.0580.*

Ioffe, S. and Szegedy, C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv Preprint arXiv:1502.03167*.

Kasowski, M., *et al.* (2010) Variation in transcription factor binding among humans. *Science*, **328**(5975), 232-235.

Kelley, D.R., *et al.* (2016) Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Res.*, **26**(7), 990-999.

Krizhevsky, A., *et al.* (2012) Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097-1105.

Lee, D. (2016) LS-GKM: a new gkm-SVM for large-scale datasets. *Bioinformatics*, **32**(14), 2196-2198.

Liu, F., *et al.* (2016) PEDLA: predicting enhancers with a deep learning-based algorithmic framework. *Sci. Rep.*, **6**, 28517.

Lobo, J.M., *et al.* (2008) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecol. and Biogeogr.*, **17**(2), 145-151.

Mercer, T.R., *et al.* (2009) Long non-coding RNAs: insights into functions. *Nat. Rev. Genet.*, **10**(3), 155-159.

Park, P.J. (2009) ChIP–seq: advantages and challenges of a maturing technology. *Nat. Rev. Genet.*, **10**(10), 669-680.

Pelossof, R., *et al.* (2015) Affinity regression predicts the recognition code of nucleic acid-binding proteins. *Nat. Biotechnol.*, **33**(12), 1242-1249.

Qin, Q. and Feng, J. (2017) Imputation for transcription factor binding predictions based on deep learning. *PLoS Comput. Biol.*, **13**(2), e1005403.

Sandelin, A., *et al.* (2004) JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Res.*, **32**, D91-94.

Smyth, D.J., *et al.* (2008) Shared and distinct genetic variants in type 1 diabetes and celiac disease. *New. Engl. J. Med.*, **359**(26), 2767-2777.

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

Wang, J., *et al.* (2012) Sequence features and chromatin structure around the genomic regions bound by 119 human transcription factors. *Genome Res.*, **22**(9), 1798-1812.

Zhou, J. and Troyanskaya, O.G. (2015) Predicting effects of noncoding variants with deep learning-based sequence model. *Nat. Methods*, **12**(10), 931-934.