Predicting Enhancer-Promoter Interaction from Genomic Sequence with Deep Neural Networks

Shashank Singh[1], Yang Yang[2], Barnabás Póczos[1], and Jian Ma[2,*]

[1]Machine Learning Department
[2]Computational Biology Department
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[*]Corresponding author: jianma@cs.cmu.edu

Abstract

In the human genome, distal enhancers are involved in regulating target genes through proximal promoters by forming enhancer-promoter interactions. However, although recently developed high-throughput experimental approaches have allowed us to recognize potential enhancer-promoter interactions genome-wide, it is still largely unknown whether there are sequence-level instructions encoded in our genome that help govern such interactions. Here we report a new computational method (named "SPEID") using deep learning models to predict enhancer-promoter interactions based on sequence-based features only, when the locations of putative enhancers and promoters in a particular cell type are given. Our results across six different cell types demonstrate that SPEID is effective in predicting enhancer-promoter interactions as compared to state-of-the-art methods that use non-sequence features from functional genomic signals. This work shows for the first time that sequence-based features alone can reliably predict enhancer-promoter interactions genome-wide, which provides important insights into the sequence determinants for long-range gene regulation.

1 Introduction

Our understanding of how the human genome regulates complex cellular functions in a living organism is still primitive. A critical challenge is to fundamentally decode the instructions encoded in our genome that govern genome organization and function. Such knowledge can in turn greatly enhance our ability to understand disease genomes. One particular aspect that we still know little about is the high-order organization of the human genome in the cell nucleus. The genome in each human cell contains approximately 6 feet long DNA being tightly folded and packaged into a nucleus with about 5μm diameter. Intriguingly, this packaging is highly organized and tightly controlled [30]. Any disruption and perturbation of the organization may lead to disease. Recent development of new high-throughput whole-genome mapping approaches such as Hi-C [22] and ChIA-PET [7, 32] has allowed us to identify genome-wide chromatin organization and interactions comprehensively. We now know that the global genome organization is more complex than previously thought, in particular, in regards to enhancer-promoter interactions. Distal regulatory enhancer elements in the genome can interact with proximal promoter regions to regulate the target gene's expression, and mutations that change such interactions will cause the target gene to be dysregulated [6, 11, 34]. In mammalian and vertebrate genomes, the promoter regions of the gene and their distal regulatory enhancers may be millions of base-pairs away from each other; and a promoter may not interact with its closest enhancer. Indeed, the mappings of global chromatin interaction based on Hi-C and ChIA-PET have shown that a significant proportion of enhancer elements skip nearby genes and interact with promoters further away in the genome by forming long-range chromatin loops [19, 29]. However, the principles encoded at the genomic sequence level underlying such organization and chromatin interaction are poorly understood.

In this work, we focus on determining whether there are sequence features encoded in the genome within enhancer elements and promoter elements that govern enhancer-promoter interactions (EPI). Although certain sequence features (e.g., CTCF binding motifs [26]) are known to mediate chromatin loops, it remains largely elusive whether and what information encoded in the genome sequence contains important instructions for forming EPI. There exists some very recent work on predicting EPI based on functional genomic features [27, 33]. In Roy et al. [27], a method called RIPPLE was developed using a combination of random forests and group LASSO in a multi-task learning framework to predict EPIs in multiple cell lines, using DNase-seq, histone marks, transcription factor (TF) ChIP-seq, and RNA-seq data as input features. Whalen et al. [33] developed TargetFinder based on boosted trees to predict EPI using DNase-seq, DNA methylation, TF ChIP-seq, histone marks, CAGE, and gene expression data. The principle of using functional genomic signals as features in both RIPPLE and TargetFinder is similar. However, the evaluation method in TargetFinder uses non-interacting enhancer-promoter pairs as negative samples, which is more reasonable for the purpose of assessing EPI predictions, but in RIPPLE, randomly sampled sequences are used as a negative dataset. We therefore directly compared with TargetFinder in our work. Nevertheless, from RIPPLE and TargetFinder, it is clear that signals from functional genomic data are informative to computationally distinguish EPIs from non-interacting enhancer-promoter pairs. These studies also suggest that important proteins and chemical modifications that may be involved in mediating the chromatin loops for EPIs can be recognized. However, it remains unclear whether the information in genome sequences within enhancers and promoters is sufficient to distinguish EPIs. Indeed, no algorithm currently exists to predict EPI using sequence-level signatures only. In this work, we want to answer the following question: *if we are only given the locations of putative enhancers and promoters in a particular cell type, can we train a predictive model to identify EPIs directly from the genomic sequences without using other functional genomic signals?* We address this question by developing a deep learning model.

In the past year, there have been many deep learning applications to regulatory genomics [1, 15, 21, 24, 25, 35]. The deep learning framework has the advantage of automatically extracting useful features from the genome sequence and can capture non-linear dependencies in the sequence to predict specific functional annotations [2]. However, three-dimensional genome organization and high-order chromatin interaction of functional elements remain an unexplored area for deep learning models.

In this paper, we develop, to the best of our knowledge, the first deep learning architecture for predicting EPIs using only sequence-based features, which in turn demonstrates that the principles of regulating EPI may be largely encoded in the genome sequences within enhancer and promoter elements. We call our model SPEID (Sequence-based Promoter-Enhancer Interaction with Deep learning; pronounced “speed”). Given the location of putative enhancers and promoters (that are largely cell-type specific) in a particular cell type, SPEID can effectively predict EPI in that cell type using sequence-based features extracted from the given enhancers and promoters. In six different cell lines, we show that SPEID achieved competitive results as compared to the state-of-the-art TargetFinder that uses non-sequence features from functional genomic signals. We believe that SPEID has the potential to become a generic model to allow us to better understand sequence level mechanistic instructions encoded in our genome that determine long-range gene regulation in different cell types.

2 Methods

2.1 Model framework of SPEID

Our SPEID model is illustrated in Figure 1. The main layers of the network are pairs of layers for convolution, activation, and max-pool layers, respectively, together with a single recurrent long short-term memory (LSTM) layer, and a dense layer.

Input, convolution, and max-pooling

The first layers of the network are responsible for learning informative subsequence features of the inputs. Because informative subsequence features may differ between enhancers and promoters, we train separate branches for each. These features might include, for example, TF protein binding motifs and other sequence-based signals. Each branch consists of a *convolution layer* and a rectified linear unit (ReLU) *activation layer* [9],

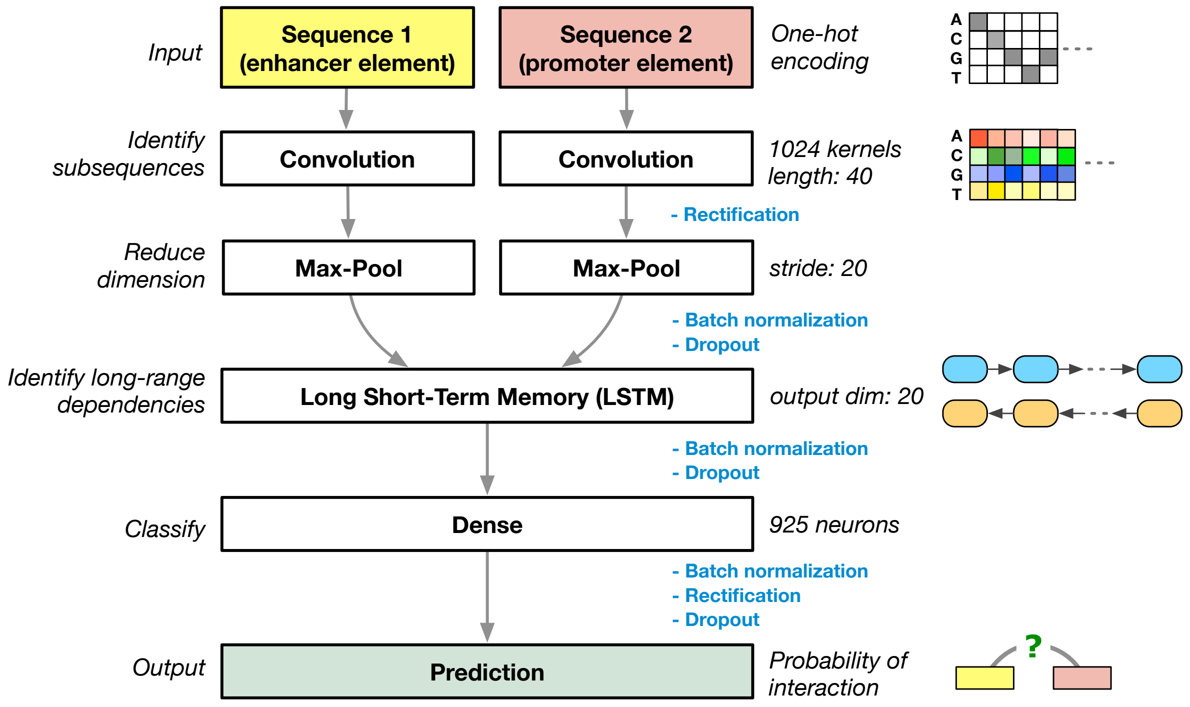

Figure 1: Diagram of our deep learning model SPEID to predict enhancer-promoter interactions based on sequences only. Key steps involving rectification, batch normalization, and dropout are annotated. Note that the final output step is essentially a logistic regression in SPEID which provides a probability to indicate whether the input enhancer element and promoter element would interact.

which together extract subsequence features from the input, and a *pooling layer*, which reduces dimensionality. The convolution layer consists of a large number (1024) of 'kernels', 4×40 signed weight matrices which are convolved with the input sequence to output a sequence of 'scores', indicating how well the kernel matches with each 40bp window of the input sequence at each possible offset. More precisely, each kernel is a matrix $K \in \mathbb{R}^{4\times 40}$, and, for each one-hot encoded input matrix $X \in \{0,1\}^{4\times L}$ and each offset $\ell \in \{0, ..., L-40\}$ (where L is the length of the input sequence), the convolution layer outputs the matrix inner product

$$C_\ell = \sum_{b=1}^{4} \sum_{j=1}^{40} K_{b,j} X_{b,\ell+j},$$

between K and the 40bp submatrix of X at offset ℓ. A higher (more positive) C_ℓ indicates a better match between K and the ℓ^{th} offset subsequence of the input.

C_ℓ is then passed through a ReLU activation $R(x) = \max\{0, x\}$, which propagates positive outputs (i.e., sequence 'matches') from the convolution layer, while eliminating negative outputs (i.e., 'non-matches'). Of the various non-linearities that can be used in deep learning models, ReLU activations are the most popular, due to their computational efficiency, and because they naturally sparsify the output of the convolution layer to only include positive matches [18].

Max-pooling then reduces the output of the convolution/activation layer by propagating only the largest output of each kernel within each 'stride' (i.e., 20 bp window), effectively outputting the 'best' alignment of each kernel within each stride. Specifically, it reduces the sequence $R(C_1), R(C_2), ..., R(C_{L-40})$ of length $L-40$ to the subsampled sequence

$$\max_{\ell\in\{1,...,20\}} R(C_\ell), \max_{\ell\in\{21,...,40\}} R(C_\ell), ..., \max_{\ell\in\{L-59,...,L-40\}} R(C_\ell),$$

of length $(L-40)/20$. Pooling is especially important in EPI prediction because of the long input sequence

(5kbp, as compared to 1kbp used when predicting sequence variant function [25, 35]). (Parameters: Number of Kernels: 1024, Filter Length: 40, L_2 penalty weight: 10^{-5}, Pool Length: 20, Stride: 20)

Before feeding into the next layer, the enhancer and promoter branches are concatenated into a single output. The remaining layers of the network act jointly on this concatenation, rather than as disjoint pairs of layers, as in the previous layers (see Figure 1).

LSTM, dense layer, and final output

The next layer is a *recurrent long short-term memory (LSTM) layer* [13], responsible for identifying informative combinations of the extracted subsequence features, across the extent of the sequence. The internal mechanism of an LSTM is fairly complex, and we refer to [10] for a detailed explanation of the particular LSTM implementation we use. As a brief intuition, the LSTM outputs a low-dimensional weighted linear projection of the input, and, as the LSTM sweeps across each element of the input sequence, it chooses, based on previous inputs, the current input, and weights learned by the model, to add or exclude each feature in this lower dimensional representation. This layer is bidirectional; that is it sweeps from both left to right and right to left, and the outputs of each direction are concatenated for a total output dimension of 200.

The final *dense layer* is simply an array of 925 hidden units with nonlinear (ReLU) activations feeding into a single sigmoid (i.e., logistic regression) unit that predicts the probability of an interaction. That is, if $y \in \mathbb{R}^{200}$ denotes the output of the LSTM layer, then then final output is a predicted interaction probability $S(v^T R(W * y)) \in (0, 1)$, where $W \in \mathbb{R}^{925\times200}$ and $v \in \mathbb{R}^{200}$ are learned weights, R denotes ReLU activation (applied to each component of Wy), and $S(t) = \frac{1}{1+e^{-t}}$ denotes the sigmoid function.

Similar (albeit simpler) architectures have been used for the related problem of predicting function of non-coding sequence variants [25, 35]. In fact, our use of a recurrent LSTM layer rather than a hierarchy of convolutional/max-pooling layers is inspired by the architecture of DanQ [25], which suggests that the LSTM is better able to model a "regulatory grammar" by incorporating long range dependencies between subsequences identified by the convolution layer. However, our method solves a fundamentally different problem – predicting interactions between sequences rather than predicting annotations from a single sequence. Hence, our model has a branched architecture, which takes two inputs and produces a single classification, rather than a sequential architecture. Because the data for this problem are far sparser, we also require a more careful training procedure, as detailed in the next section. There are also several finer distinctions between the models, such as our use of batch normalization to accelerate training and weight regularization to improve generalization.

Other model and implementation details

As suggested by [25], we inject some prior knowledge by initializing about half (525 of 1024) of the convolutional kernels with known motifs from the JASPAR database [23] Specifically, recall that each kernel is a 4×40 weight matrix. To initialize a kernel with a JASPAR motif of length $\ell < 40$bp, we choose a uniformly random length-ℓ subsequence of the kernel and initialize it with the position weight matrix from JASPAR. We then initialize the remainder of the kernel with independent standard uniform random values. Note that convolutional kernels are model parameters to be learned from the data (via the backpropagation algorithm [28]), and the model is free to retain, modify, or discard these initial values based on whether they are useful for prediction. Also, we only do this for the prediction; when analyzing the learned convolutional features (in Section 3.3), we initialize all kernels, and indeed all model weights, with uniform random numbers, scaled by the sizes of the input and output layers as suggested by [8]. It is important to note that initializing using JASPAR motifs does not provide bias towards known TF motifs for our prediction. We found that the randomly initialized model achieves quite similar performance (within 3% AUROC and AUPR, across all cell lines).

We implemented our deep learning model using Keras 1.1.0 [4]. The model was trained in minibatches of 100 samples by back-propagation, using binary cross-entropy loss, minimized by Adam [16] with a learning rate of 10^{-5}. The pretraining and retraining phases described in Section 2.3 lasted 32 epochs and 80 epochs, respectively. The training time was linear in the sample size for each cell line, taking, for example, 11 and 6 hours for pretraining and retraining phases, respectively, on K562 data, on an NVIDIA GTX 1080 GPU.

2.2 Addressing potential overfitting

When training very large models such as deep networks, potential overfitting is a concern. This is particularly relevant because our datasets are small ($< 10^4$ positive samples per cell line) compared to the massive data sets often used to train deep networks (e.g., with imaging or text data). Thus, we employ multiple techniques to prevent or mitigate overfitting, both within the deep learning model, and in our training and evaluation procedures.

Firstly, note that we provide results on 6 independent data sets from different cell lines. While we experimented with multiple deep network models, utilizing different layers and training hyperparameters, we did this based on only test results from K562. In particular, results on the remaining 5 cell lines are independent of model selection. Since prediction results on the remaining cell lines are similar to those on K562, we conclude that overfitting due to model selection is not a concern; i.e., the selected model generalizes well to new data. Second, by randomly shifting positive inputs, our data augmentation procedure described in Section 2.3 specifically prevents the model from overfitting to any region of the input. Finally, the deep network model itself incorporates three tools to reduce overfitting: batch normalization, dropout, and L_2 regularization.

Batch normalization [14] is the process of linearly normalizing the outputs of neurons on each training batch to have sample mean 0 and standard deviation 1. That is, if the i^{th} batch consists of 100 samples for which a particular neuron gives outputs $N_{i,1}, ..., N_{i,100}$, then we replace these outputs with normalized outputs

$$\frac{N_{i,1} - \bar{N}_i}{\sigma_i}, ..., \frac{N_{i,100} - \bar{N}_i}{\sigma_i}, \quad \text{where} \quad \bar{N}_i = \frac{1}{n}\sum_{j=1}^{100} N_{i,j} \quad \text{and} \quad \sigma_i^2 = \frac{1}{n-1}\sum_{j=1}^{100}\left(N_{i,j} - \bar{N}_i\right)^2$$

are the sample mean and sample variance, respectively. Batch normalization combats overfitting by limiting the range of non-linearities (in our case, ReLU function) in the network, and also accelerates training (i.e., reduces the number of epochs till convergence) by restricting the input space of downstream neurons. We batch-normalize the outputs of 4 layers in the network: the max-pooling layer, the LSTM layer, the dense layer (i.e., before the ReLU activation), and the ReLU activation (i.e., before the final sigmoid classifier). Note that, during prediction, batch means and variances are replaced with population means and variances, which are computed while training over all batches.

Dropout, a common regularization technique in neural networks, refers to randomly 'dropping' (i.e., setting to zero) the output of a neuron with some fixed probability p. That is, for each sample i and each neuron j, we sample a Bernoulli(p) random variable $D_{i,j}$ and replace the output $N_{i,j}$ with $(1 - D_{i,j})N_{i,j}$. Applying dropout to a layer L_i prevents the subsequent layer L_{i+1} from overfitting to any subset of neurons L_i in intermediate layers, and there by promotes a distributed representation, which can be thought of as *model averaging* (i.e., learning the average of many models, as in ensemble techniques). We apply dropout with $p = 0.5$ to the outputs of 3 layers: the max-pooling layer, the LSTM, and the dense layer (dropout is always applied after batch normalization). Note that dropout is only applied during training; all neurons' outputs are used during testing.

Finally, we apply an L_2-norm penality on the matrix of weights of the dense layer. While a similar regularization penalty is often applied to the convolutional kernels, this interferes with accurately learning sequence motifs, so we omit this.

2.3 Training procedure

Recall that our data set is highly imbalanced – there are many more negative (non-interacting) pairs than positive (interacting) pairs. In each cell line, there are typically 20 times more negative samples than positive samples. To combat the difficulty of learning highly imbalanced classes, we utilize a two-stage training procedure that involves pre-training on a data set balanced with data augmentation, followed by training on the original data.

Pre-training with data augmentation

Data augmentation is commonly used as an alternative to re-weighting data when training deep learning models on highly imbalanced classes. For example, image data is often augmented with random translations, scalings, and rotations of the original data. In our case, because enhancers and promoters are typically smaller than the fixed window size we use as input (as described in Section 3.1), the labels are invariant to small shifts of the input sequence, as long as the enhancer or promoter remains within this window. By randomly shifting each positive promoter and enhancer within this window, we generated "new" positive samples. We did this 20 times with each positive sample, effectively balancing the positive and negative classes. In addition to balancing class sizes, this data augmentation has the additional benefit of promoting translation invariance in our model, preventing it from overfitting to any particular region of the input sequence.

Imbalanced training

Data augmentation results in a consistent training procedure for the network, allowing the convolutional layers to identify informative subsequence features and the recurrent layer to identify long-range dependencies between these features. However, in typical applications of predicting interactions, classes are, as in our original data, highly imbalanced. In these contexts, naively using the network trained on augmented data results in a very high false positive rate.

Fortunately, this has relatively little to do with the convolutional and recurrent layers of the network, which correctly learn features that distinguish positive and negative samples, and this issue is largely due to the dense layer, which performs prediction based on these features. Hence, to correct for this, we only retrain the dense layer. We do this by "freezing" the lower layers of the network (i.e., setting the learning rate to 0), and then continuing to train the network as usual on the subset of the original imbalanced data that was used to generate the augmented data.

Summary of training procedure

The following procedure is repeated independently for each of the 6 cell lines we study:

1. Begin with an imbalanced data set A.
2. Split A uniformly at random into a training set B (90% of A) and a test set C (10% of A).
3. Augment positive samples in B to produce a balanced data set D.
4. Train the model on D, using a small (10%) subset for model validation.
5. Freeze the convolution and recurrent layers of the model.
6. Continue training the dense layer of the model on B.
7. Evaluate on C.

3 Results

3.1 Data preparation

We utilized the EPI data sets previously used in TargetFinder [33] for our model training and evaluation so that we can also directly compare with TargetFinder. The data include six cell lines (GM12878, HeLa-S3, HUVEC, IMR90, K562, and NHEK). Cell-line specific active enhancers and promoters were identified using annotations from the ENCODE Project [5] and Roadmap Epigenomics Project [3]. The locations of these putative enhancers and promoters are the input for SPEID for each cell line. The data for each cell line consist of enhancer-promoter pairs which are annotated as positive (interacting) or negative (non-interacting) using high-resolution genome-wide measurements of chromatin contacts in each cell line based on Hi-C [26], as used in [33]. 20 negative pairs were sampled per positive pair, under constraints on the genomic distance between the paired enhancer and promoter as described in [33], such that positive and negative pairs had similar enhancer-promoter distance distributions. The resultant datasets are heavily imbalanced, in accordance with the fact that enhancer/promoter interactions are far fewer than non-interactions.

To address the problem of class imbalance, we applied data augmentation to the positive pairs, as motivated in Section 2.3. The original annotated enhancers in the datasets of each cell type are mostly only a few hundred base pairs (bp) in length, with average length varying from 340 bp to 720 bp across the six cell types. We extended the enhancers to be 3 kbp in length by including adjustable flanking regions, both for augmentation of positive samples and for more informative feature extraction with the use of the surrounding sequence context. The enhancers are fitted to a uniform length with the extensions, as sequences of fixed sizes are needed as input to our model. The original annotated promoters are mostly 1-2 kbp in length, with average varying from 450 bp to 1.96 kbp across cell types. Promoters are similarly fitted to a fixed window size of 2 kbp. If the original region is shorter than 3 kbp (for enhancer) or 2 kbp (for promoter), random shifting of the flanking regions is performed to get multiple samples. If it is longer than 3 kbp or 2 kbp, segments of the fixed length are randomly sampled from the original sequence.

For negative pairs, enhancers and promoters are also fixed to the window sizes of 3 kbp and 2 kbp, respectively, in the similar approach performed over the positive pairs, but without augmentation. The numbers of positive pairs, augmented positive pairs, and negative pairs on each cell line and the combined cell lines are listed in Table 1. We convert the genomic sequence to a 4×3000 matrix (for enhancer) and 4×2000 matrix (for promoter) as inputs to our model with a one-hot encoding (e.g., 'A' is $(1, 0, 0, 0)^T$, 'G' is $(0, 1, 0, 0)^T$, 'C' is $(0, 0, 1, 0)^T$, and 'T' is $(0, 0, 0, 1)^T$).

Cell Line	Positive Pairs	Augmented Positive Pairs	Negative Pairs
GM12878	2,113	42,260	42,200
HeLa-S3	1,740	34,800	34,800
HUVEC	1,524	30,480	30,400
IMR90	1,254	25,080	25,000
K562	1,977	39,540	39,500
NHEK	1,291	25,820	25,600
Total	9,899	197,980	197,500

Table 1: Positive sample, augmented positive sample, and negative sample counts, for each cell line.

3.2 Evaluation results as compared to TargetFinder

We compared our prediction results to the state-of-the-art model, TargetFinder [33], which uses boosted trees to predict EPI based on numerous functional genomic signals and annotations, including DNase-seq, DNA methylation, TF ChIP-seq, histone marks, CAGE, and gene expression data. TargetFinder has 3 variants, which use features from different regions: Enhancer/Promoter (E/P) uses only annotations within the enhancer and promoter, Extended Enhancer/Promoter (EE/P) additionally uses annotations within an extended 3kbp flanking region around each enhancer, and Enhancer/Promoter/Window (E/P/W) additionally uses annotations in the region between the enhancer and promoter. Note that in SPEID, our input sequences only include the surrounding sequences of the enhancer and promoter (as we discussed above).

Table 2 and Table 3 show comparisons of results between our SPEID method and different TargetFinder models, on each of 6 different cell types. As described in Section 2.3, SPEID results are reported on a held-out subset of 10% of the original data, separate from the subset used for model validation. Results across independent cell lines suggest that the comparison is fairly reliable, even without the computational expense of training the model several times for cross-validation. Due to high class imbalance, we report F_1 score (harmonic mean of precision and recall) in Table 2, rather than accuracy. We found that, although results vary across different cell lines, the performance of SPEID is comparable to that of TargetFinder's best model (Table 2, Table 3).

Table 4 and Table 5 show the results from SPEID in terms of AUROC and AUPR when training and testing are on different cell lines. As expected, within-cell-line prediction achieved the best performance, while cross-cell-line prediction had much worse performance, consistent with the fact that EPIs are quite cell-type specific.

If SPEID is given the cell-type specific locations of enhancers and promoters, then it can effectively predict EPI using sequence-based features *specific to EPI in that cell type*. However, SPEID depends heavily on sequence features that are cell-type specific, and is unable to automatically capture relevant sequence features operating cross-cell-type. Taken together, our results suggest that sequence-based features do have important information that can determine EPI, and, if we are given the locations of enhancers and promoters for a particular cell type, our SPEID model can effectively predict EPI using sequence features only. As more positive EPI samples and data in additional cell types becomes available, more work is needed to determine what, if any, sequence features mediate EPI consistently across cell types.

	Cell Type					
Model	GM12878	HeLa-S3	HUVEC	IMR90	K562	NHEK
SPEID	**0.85**	0.81	0.75	**0.78**	**0.85**	**0.94**
TargetFinder (E/P)	0.59	0.61	0.48	0.48	0.61	0.83
TargetFinder (EE/P)	0.84	0.83	0.71	0.83	0.81	0.83
TargetFinder (E/P/W)	0.81	**0.87**	**0.77**	**0.78**	**0.85**	0.90

Table 2: F_1 scores of different EPI prediction methods for each cell line.

	Cell Type					
Model	GM12878	HeLa-S3	HUVEC	IMR90	K562	NHEK
SPEID	**0.87**	**0.87**	**0.88**	0.87	**0.90**	0.88
TargetFinder (E/P)	0.76	0.59	0.69	0.69	0.77	0.76
TargetFinder (EE/P)	0.90	0.82	0.81	**0.88**	0.87	0.89
TargetFinder (E/P/W)	**0.87**	0.82	0.84	0.86	**0.90**	**0.93**

Table 3: Area under ROC curve (AUROC) for different EPI prediction methods. Note that we only show AUROC comparisons since TargetFinder did not report AUPR.

	Testing Cell Type					
Training Cell Type	GM12878	HeLa-S3	HUVEC	IMR90	K562	NHEK
GM12878	**0.87**	0.62	0.64	0.64	0.62	0.59
HeLa-S3	0.60	**0.87**	0.68	0.56	0.62	0.62
HUVEC	0.63	0.67	**0.88**	0.62	0.63	0.66
IMR90	0.62	0.63	0.64	**0.87**	0.60	0.64
K562	0.64	0.63	0.63	0.57	**0.90**	0.59
NHEK	0.58	0.65	0.66	0.56	0.59	**0.88**

Table 4: Area under ROC curve (AUROC) for SPEID when training and testing on different cell lines.

	Testing Cell Type					
Training Cell Type	GM12878	HeLa-S3	HUVEC	IMR90	K562	NHEK
GM12878	**0.49**	0.09	0.10	0.09	0.09	0.07
HeLa-S3	0.07	**0.55**	0.10	0.06	0.09	0.11
HUVEC	0.09	0.12	**0.57**	0.08	0.10	0.13
IMR90	0.09	0.08	0.10	**0.56**	0.08	0.09
K562	0.12	0.14	0.12	0.08	**0.80**	0.10
NHEK	0.07	0.11	0.12	0.08	0.09	**0.67**

Table 5: Area under precision-recall curve (AUPR) for SPEID when training and testing on different cell lines.

3.3 Analysis of convolution features

One way of understanding the features related to TF binding learned by SPEID is to compare the patterns of the convolutional kernels learned during training to known TF binding motifs, although SPEID also captures other informative features that do not match TF motifs. Note that, since we are interested in studying the subsequences that are learned by SPEID, in this section, we used a randomly initialized model (i.e., not initialized with motifs from JASPAR). Using a similar procedure as in [25] and [1], we converted each kernel into a position frequency matrix (PFM). In short, this involves reconstructing the rectified output of the convolutional layer on each input sample sequence to identify subsequence alignments that best match each kernel, and then computing PFMs from these aligned subsequences. Exact details of this procedure can be found in Supplementary Section 10.2 of [1]. We then used the motif comparison tool Tomtom 4.11.2 [12] to match these PFMs to known TF binding motifs from the HOCOMOCO Human v10 database [17], which includes 641 motifs for 601 human TFs.

Due to non-linearities in the deep network and the use of dropout during training, it is not obvious how to measure importance of specific convolutional features to prediction. Dropout encourages the model to develop redundant representations for important features, so that they are consistently available. As a result, we cannot measure importance of a feature in terms of the change in prediction performance when holding that feature out. For the same reason, however, one measure of a feature's importance is the redundancy of that feature's representation in the model. Specifically, when using a dropout probability of 50%, the probability of a convolutional feature being available to the model is $1 - 2^{-r}$, where r is the number of copies of that convolutional feature, so that r will be larger for more important features. As an example, the most frequently observed motif was the binding pattern of MAZ, which matched with 28 promoter convolution kernels. MAZ was similarly, reported as "of high importance" in promoter regions by TargetFinder.

With this reasoning, to prune the number of motif matches, we report TFs which independently matched with at least 3 kernels in SPEID with an E-value of less than $E < 0.5$ according to Tomtom. Table 6 gives, for each cell line, the number of motifs thus identified by SPEID, as well as the numbers of features found to be in the top 50% of importance by TargetFinder. Note that it is difficult to directly compare the motifs discovered by SPEID with the features found important by TargetFinder. Firstly, the latter used many features such as histone marks and gene expression data that lack corresponding motifs. Even among TF features, many do not have corresponding motifs in the HOCOMOCO database. Secondly, TargetFinder focused on TF ChIP-seq signals as features, not only in the enhancer and promoter regions, but also in the window region between them, while SPEID only uses sequence features from the input enhancer and promoter sequences.

Cell Line	Predicted important in both	Only in SPEID	Only in TargetFinder
GM12878	22	9	53
HeLa-S3	13	15	37
HUVEC	1	14	7
IMR90	4	31	16
K562	27	26	85
NHEK	0	16	5

Table 6: Number of potentially important TFs in enhancers involved in EPIs identified by SPEID, TargetFinder, or both. Here we consider the top 50% most informative features from TargetFinder as important. We found that the two methods have the largest overlap in GM12878 and K562 cell lines, partly because TargetFinder used many more TF ChIP-seq signals for these two cell lines.

Among the features that can reasonably be compared, the results suggest many commonalities between the findings of SPEID and TargetFinder. For example, of the 27 K562 enhancer motifs we discovered with corresponding features in TargetFinder, 23 were in the 30% of features considered most important by TargetFinder. Table 7 gives TFs with motifs discovered by SPEID, in two cell lines GM12878 and K562 (the two cell types with the largest number features in TargetFinder having corresponding motif, as well as the largest EPI datasets). These include known TFs such as CTCF, as well as a number of TFs whose roles in EPI have not been well studied at present, such as SRF, JUND, SPI1, SP1, EBF1, and JUND, which were also reported in [33]. This comparison demonstrates the potential of SPEID in identifying important TFs involved in EPI.

	Potentially important TFs involved in EPI
Also in TargetFinder (GM12878)	SPI1, EBF1, SP1, IRF3, TCF12, BATF, PAX5, MEF2A, BCL11A, EGR1, SRF, IRF4, BHLHE40, PBX3, MEF2C, MAZ, NRF1, YY1, GABPA, ETS1, STAT1, NFYA
Unique in SPEID (GM12878)	CPEB1, HXC10, ARI3A, IRF1, IRF8, MNT, TBX15, TBX2, TBX5
Also in TargetFinder (K562)	CTCF, SRF, ATF3, MAZ, JUND, MEF2A, CEBPD, BHLHE40, NR2F2, EGR1, FOSL1, FOS, TAL1, JUNB, JUN, MAFK, E2F6, SP1, NFE2, NR4A1, GATA1, THAP1, SP2, RFX5, NRF1, USF2
Unique in SPEID (K562)	SP4, SP3, TFDP1, ZFX, WT1, KLF15, TBX1, ETV1, ZNF148, KLF6, HEN1, KLF14, TBX15, CLOCK, ELF2, PLAL1, PURA, ZNF740, AP2D, CPEB1, EGR2, FOXJ3, HES1, NR1I3, SREBF2, THA

Table 7: Predicted potentially important TFs in enhancers involved in EPIs from SPEID in GM12878 and K562.

4 Conclusions and Future Work

The long-range interaction between enhancer and promoter is one of the most intriguing phenomena in gene regulation. Although new high-throughput experimental approaches have provided us with tools to identify potential EPIs genome-wide, it is largely unclear whether there are sequence-level instructions already encoded in our genome that help determine EPIs. In this work, we have developed, to the best of our knowledge, the first deep learning model, SPEID, to directly tackle this question. The contribution of our method is two-fold: (1) We have shown that sequenced-based features alone can indeed effectively predict EPIs if we are given the genomic locations of putative enhancers and promoters in a particular cell type. We demonstrated that SPEID can achieve performance competitive with the state-of-the-art method TargetFinder that uses numerous functional genomic signals instead of sequence features. (2) The deep learning framework in SPEID provides a useful predictive model for studying genomic sequence-level interactions by extracting relevant sequence information, representing an important conceptual expansion of the application of deep learning models in regulatory genomics.

There are a number of directions that our method can be further improved. For example, our current ability in determining the potentially informative features remains limited. Although we were able to identify some informative TFs that may play roles in mediating EPIs in a certain cell type and that were also identified from TargetFinder, a significant proportion of sequence features from our model cannot be easily interpreted and their contributions are also hard to evaluate. A natural next step is to apply very recently developed methods such a DeepLIFT [31] and deep feature selection [20] for measuring importance of and selecting among different convolutional features, to identify those that might mediate EPI within or across cell lines. In addition, even though we successfully demonstrated that sequence features alone can effectively predict EPI, it would be important to explore the optimal combination of sequence-based features and features from functional genomic signals to achieve the strongest predictive power in a cell-type specific manner. Such an approach would be highly useful in understanding both the genetic and epigenetic mechanisms that determine EPIs and their variation across different cell types. Nevertheless, we believe our work provides a useful new framework to potentially decode important sequence determinants for long-range gene regulation.

Acknowledgments

S.S. is supported by a National Science Foundation Graduate Research Fellowship DGE-1252522. J.M. is supported by National Institutes of Health grants HG007352, DK107965, and CA182360, and National Science Foundation grants 1054309 and 1262575.

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*, 2015.

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

[3] B. E. Bernstein, J. A. Stamatoyannopoulos, J. F. Costello, B. Ren, A. Milosavljevic, A. Meissner, M. Kellis, M. A. Marra, A. L. Beaudet, J. R. Ecker, et al. The NIH roadmap epigenomics mapping consortium. *Nature biotechnology*, 28(10):1045–1048, 2010.

[4] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[5] E. P. Consortium et al. The ENCODE (ENCyclopedia of DNA elements) project. *Science*, 306(5696): 636–640, 2004.

[6] J. R. Dixon, I. Jung, S. Selvaraj, Y. Shen, J. E. Antosiewicz-Bourget, A. Y. Lee, Z. Ye, A. Kim, N. Rajagopal, W. Xie, et al. Chromatin architecture reorganization during stem cell differentiation. *Nature*, 518 (7539):331–336, 2015.

[7] M. J. Fullwood and Y. Ruan. Chip-based methods for the identification of long-range chromatin interactions. *Journal of Cellular Biochemistry*, 107(1):30–39, 2009.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pages 249–256, 2010.

[9] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, volume 15, page 275, 2011.

[10] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

[11] Y. Guo, Q. Xu, D. Canzio, J. Shou, J. Li, D. U. Gorkin, I. Jung, H. Wu, Y. Zhai, Y. Tang, et al. Crispr inversion of ctcf sites alters genome topology and enhancer/promoter function. *Cell*, 162(4):900–910, 2015.

[12] S. Gupta, J. A. Stamatoyannopoulos, T. L. Bailey, and W. S. Noble. Quantifying similarity between motifs. *Genome biology*, 8(2):1, 2007.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[15] 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*, 2016.

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

[17] I. V. Kulakovskiy, I. E. Vorontsov, I. S. Yevshin, A. V. Soboleva, A. S. Kasianov, H. Ashoor, W. Baalawi, V. B. Bajic, Y. A. Medvedeva, F. A. Kolpakov, et al. Hocomoco: expansion and enhancement of the collection of transcription factor binding sites models. *Nucleic acids research*, 44(D1):D116–D125, 2016.

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

[19] G. Li, X. Ruan, R. K. Auerbach, K. S. Sandhu, M. Zheng, P. Wang, H. M. Poh, Y. Goh, J. Lim, J. Zhang, et al. Extensive promoter-centered chromatin interactions provide a topological basis for transcription regulation. *Cell*, 148(1):84–98, 2012.

[20] Y. Li, C.-Y. Chen, and W. W. Wasserman. Deep feature selection: Theory and application to identify enhancers and promoters. In *Research in Computational Molecular Biology*, pages 205–217. Springer,

2015.

[21] Y. Li, W. Shi, and W. W. Wasserman. Genome-wide prediction of cis-regulatory regions using supervised deep learning methods. *bioRxiv*, page 041616, 2016.

[22] E. Lieberman-Aiden, N. L. Van Berkum, L. Williams, M. Imakaev, T. Ragoczy, A. Telling, I. Amit, B. R. Lajoie, P. J. Sabo, M. O. Dorschner, et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009.

[23] 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*, page gkv1176, 2015.

[24] Y. Park and M. Kellis. Deep learning for regulatory genomics. *Nature Biotechnology*, 33(8):825–826, 2015.

[25] D. Quang and X. Xie. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic acids research*, page gkw226, 2016.

[26] S. S. Rao, M. H. Huntley, N. C. Durand, E. K. Stamenova, I. D. Bochkov, J. T. Robinson, A. L. Sanborn, I. Machol, A. D. Omer, E. S. Lander, et al. A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, 2014.

[27] S. Roy, A. F. Siahpirani, D. Chasman, S. Knaack, F. Ay, R. Stewart, M. Wilson, and R. Sridharan. A predictive modeling approach for cell line-specific long-range regulatory interactions. *Nucleic acids research*, 43(18):8694–8712, 2015.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[29] A. Sanyal, B. R. Lajoie, G. Jain, and J. Dekker. The long-range interaction landscape of gene promoters. *Nature*, 489(7414):109–113, 2012.

[30] T. Sexton and G. Cavalli. The role of chromosome domains in shaping the functional genome. *Cell*, 160 (6):1049–1059, 2015.

[31] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[32] Z. Tang, O. J. Luo, X. Li, M. Zheng, J. J. Zhu, P. Szalaj, P. Trzaskoma, A. Magalska, J. Wlodarczyk, B. Ruszczycki, et al. Ctcf-mediated human 3d genome architecture reveals chromatin topology for transcription. *Cell*, 163(7):1611–1627, 2015.

[33] S. Whalen, R. M. Truty, and K. S. Pollard. Enhancer-promoter interactions are encoded by complex genomic signatures on looping chromatin. *Nature Genetics*, 2016.

[34] Y. Zhang, C.-H. Wong, R. Y. Birnbaum, G. Li, R. Favaro, C. Y. Ngan, J. Lim, E. Tai, H. M. Poh, E. Wong, et al. Chromatin connectivity maps reveal dynamic promoter-enhancer long-range associations. *Nature*, 504(7479):306–310, 2013.

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