Supplementary materials for Chromatin Accessibility Prediction via Convolutional Long Short-Term Memory Networks with *k*-mer Embedding

Xu Min[1], Wanwen Zeng[2], Ning Chen[1], Ting Chen[2,*] and Rui Jiang[2,*]

MOE Key Laboratory of Bioinformatics; Bioinformatics Division and Center for Synthetic & Systems Biology, TNLIST; [1] Department of Computer Science and Technology; State Key Lab of Intelligent Technology and Systems; [2] Department of Automation; Tsinghua University, Beijing 100084, China.

[*] Corresponding author. Email: tingchen@tsinghua.edu.cn, ruijiang@tsinghua.edu.cn

Supplementary Figures

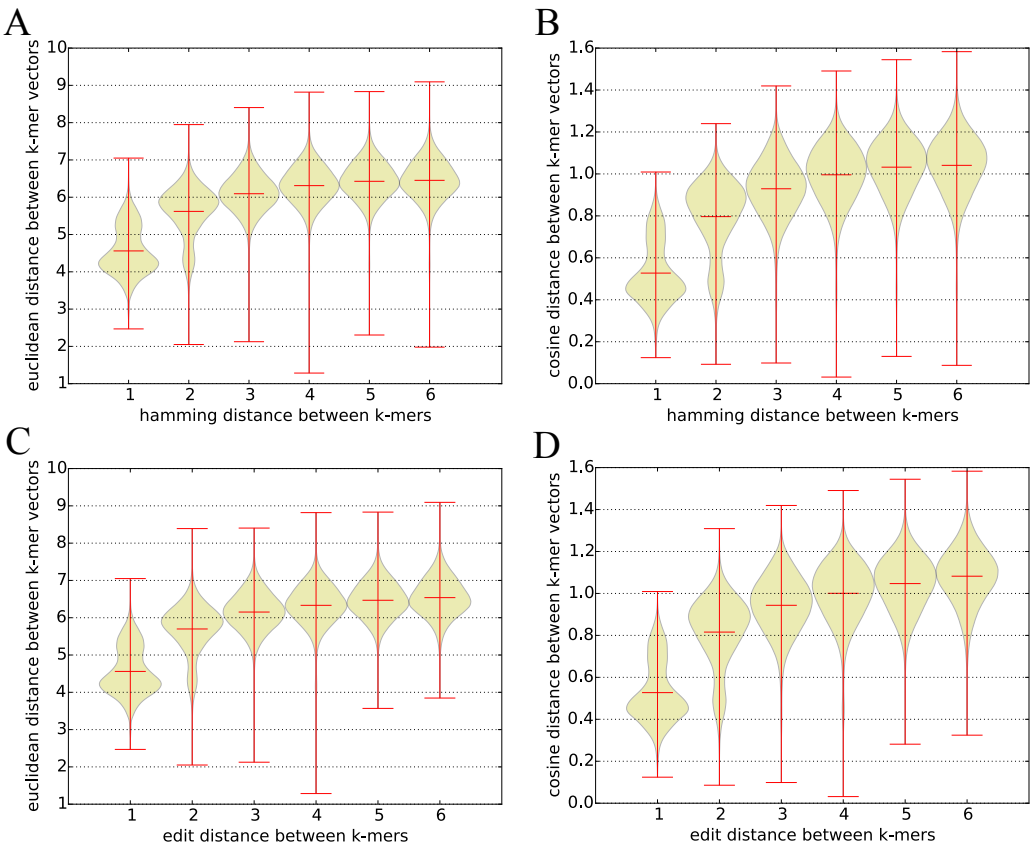

Fig. S1 Relationship between *k*-mer distance and *k*-mer vector distance. To explore how the distance between *k*-mers is related to the distance between *k*-mers themselves, we compute the pairwise Euclidean distance and cosine distance of the 4096 *k*-mer vectors trained on the MCF-7 dataset, as well as the pairwise Hamming distance and edit distance of those 4096 *k*-mers themselves. There are totally 4096*4095/2=8386560 pairs of *k*-mers, and the Hamming distance has only six possible values, i.e., [1, 2, 3, 4, 5, 6], while the edit distance also has only six possible values, i.e., [1, 2, 3, 4, 5, 6]. We split all the 8386560 pairs into six groups according to their *k*-mer distance, and then look into each group to see the distribution of their corresponding *k*-mer vector distances as is shown in violin plot. In (A) and (B), we use hamming distance as *k*-mer distance, while in (C) and (D) we use edit distance instead. In (A) and (C), Euclidean distance is used as distance between *k*-mer vectors, while in (B) and (D) cosine distance is used instead. We find that, consistently in four cases, the mean distance between *k*-mer vectors is monotonically increasing with the distance between *k*-mers.

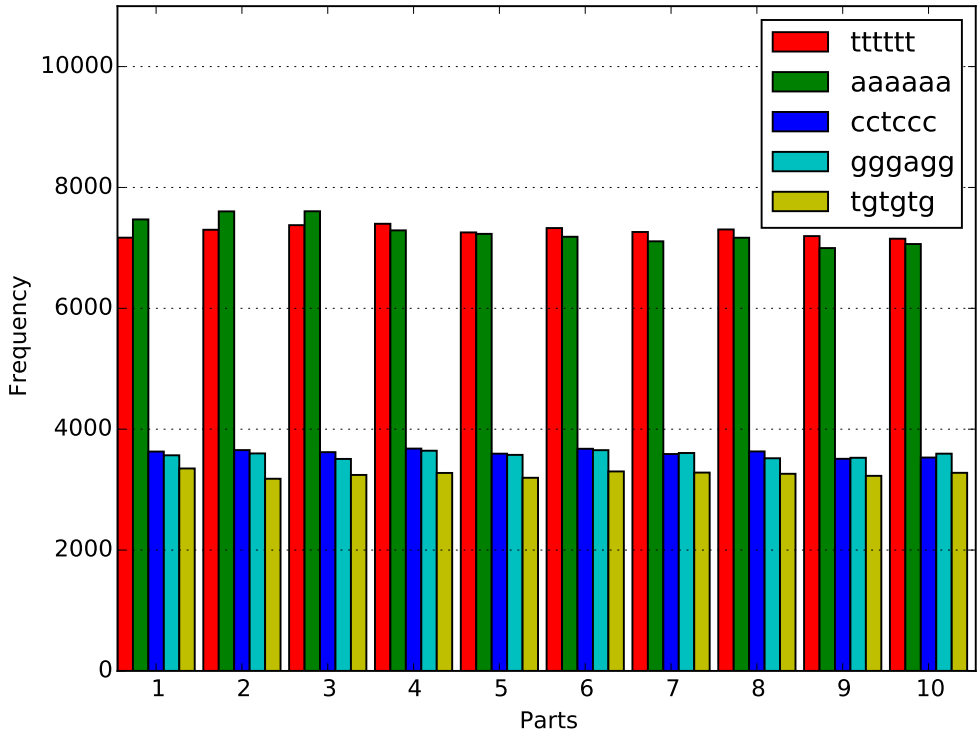

Fig. S2 *k*-mers counts in different parts of chromatin accessible regions. In order to investigate whether those enriched motifs or *k*-mers have localization preference in chromatin accessible regions, we split each chromatin accessible region into 10 equal parts, and count the frequency of these top enriched *k*-mers in each part. In GM12878 dataset, we pick out the top five enriched *k*-mers, including 'ttttt', 'aaaaaa', 'cctccc', 'gggagg', and 'tgtgtg', and look at their counts in different parts of chromatin accessible regions. From our result, there is no explicit evidence showing that enriched *k*-mers have location preference, e.g., prefer to locate in middle regions, in chromatin accessible regions.

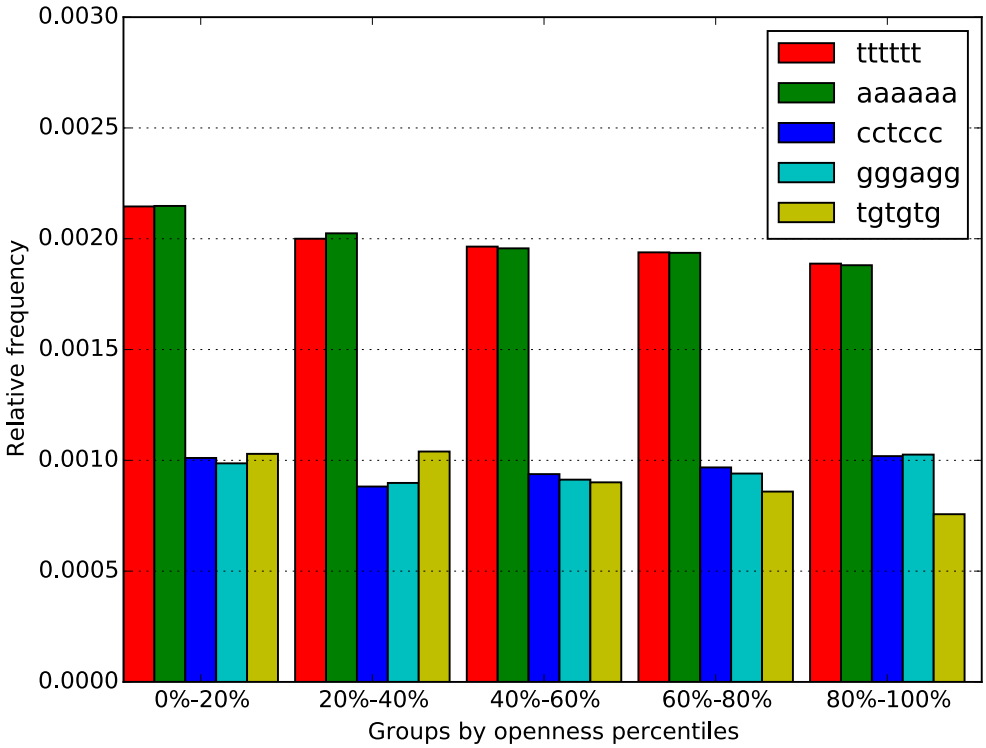

Fig. S3 *k*-mer relative frequency in different groups of chromatin accessible regions. To explore whether the occurrence of these *k*-mers has certain relationship with the openness of the regions, we divide all these chromatin accessible regions according to their openness percentile values, e.g. 0%, 20%, 40%, 60%, 80% and 100% percentiles, into 5 groups, and count the relative abundance of these top enriched *k*-mers in each group of regions. The relative abundance is defined as the *k*-mer frequency divided by the total number of *k*-mers. In GM12878 dataset, the 0%, 20%, 40%, 60%, 80% and 100% percentiles of openness values are 0.75, 3.47, 4.91, 6.83, 10.36, and 4231.13 respectively, and we use them as thresholds to divide sequences into 5 groups, each of which has 24484 or 24485 sequences. We demonstrate the relative frequency of the top five enriched *k*-mers, including 'ttttt', 'aaaaaa', 'cctccc', 'gggagg', and 'tgtgtg' in the five groups. We find that with the openness value increasing, the relative frequency of the top two *k*-mers slowly decreases. We also find that *k*-mer 'gggagg' is most enriched in the highly accessible regions, while 'tgtgtg' is least enriched in the highly accessible regions.

Supplementary Tables

Table S1 Data shape of each layer in our model. *(rows, cols)* means the dimension of the data.

Layer	Output shape
Embedding_1	(1000, 100)
Dropout_1	(1000, 100)
Convolution1d_1	(991, 100)
Maxpooling1d_1	(247, 100)
Dropout_2	(247, 100)
Convolution1d_2	(240, 100)
Maxpooling1d_2	(120, 100)
Dropout_3	(120, 100)
Convolution1d_3	(113, 80)
Maxpooling1d_3	(56, 80)
Dropout_4	(56, 80)
Lstm_1	(80)
Dropout_5	(80)
Dense_1	(20)
Dropout_6	(20)
Dense_2	(1)

Table S2 10-fold cross validation for our method and DeepSEA on GM12878 dataset. To further confirm the superiority of our model over DeepSEA, we carry out 10-fold cross validation experiments for both our model and DeepSEA. We find that the average auROC score for our method is 0.88146 and the standard deviation is 0.00434, while the average auROC score for DeepSEA is 0.86149 which is obviously lower, and the standard deviation is 0.01223 which is obviously larger than ours. Concretely, we perform one-sided Wilcoxon tests on the auROC and auPRC scores for the two methods against the alternative hypothesis that our method produces larger values of a criterion, respectively. The *p*-value of Wilcoxon test on auROC scores data is 0.0001028, choosing the alternative hypothesis that our auROC is greater than that of DeepSEA. Similarly, the *p*-value in auPRC case is 0.000525 which is also very small. In summary, through the 10-fold cross validation experiments, we are confirmed that our method significantly outperforms DeepSEA, more accurately in the auROC measure and also more stably. We also list 10-fold cross validation experiments on the rest five datasets in Table S3-S7.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.88413	0.88222	0.86517	0.86039
1	0.87657	0.86888	0.84461	0.83446
2	0.88554	0.88233	0.85086	0.84685
3	0.87352	0.87033	0.87892	0.87752
4	0.87684	0.87284	0.87505	0.87359
5	0.88669	0.88299	0.87547	0.87374
6	0.88222	0.88032	0.85034	0.84833
7	0.88394	0.88214	0.86190	0.85996
8	0.88204	0.87998	0.85023	0.84630
9	0.88314	0.87782	0.86238	0.85921
Mean	0.88146	0.87798	0.86149	0.85804
Standard Deviation	0.00434	0.00534	0.01223	0.01411

Table S3 10-fold cross validation for our method and DeepSAE on K562 dataset.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.88224	0.87660	0.85593	0.85086
1	0.87792	0.86957	0.86642	0.85830
2	0.88121	0.87450	0.86016	0.85266
3	0.88084	0.87115	0.86756	0.85872
4	0.88050	0.87073	0.85775	0.84789
5	0.88320	0.87689	0.85834	0.85220
6	0.88207	0.87247	0.86904	0.85950
7	0.88152	0.87293	0.85390	0.84552
8	0.87731	0.86907	0.86332	0.85680
9	0.88108	0.87364	0.86095	0.85281
Mean	0.88079	0.87275	0.86134	0.85353
Standard Deviation	0.00185	0.00271	0.00512	0.00474

Table S4 10-fold cross validation for our method and DeepSEA on MCF-7 dataset.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.92256	0.91827	0.92045	0.91655
1	0.92261	0.91885	0.91864	0.91501
2	0.92310	0.91981	0.91322	0.91029
3	0.92323	0.91812	0.91239	0.90748
4	0.91967	0.91305	0.92427	0.91892
5	0.92022	0.91626	0.91364	0.91049
6	0.91897	0.91425	0.91135	0.90723
7	0.92292	0.91973	0.91591	0.91179
8	0.92243	0.91903	0.92164	0.91827
9	0.91955	0.91410	0.92011	0.91440
Mean	0.92153	0.91715	0.91716	0.91304
Standard Deviation	0.00170	0.00253	0.00445	0.00422

Table S5 10-fold cross validation for our method and DeepSEA on HeLa-S3 dataset.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.89656	0.89189	0.88977	0.88889
1	0.90033	0.89830	0.87887	0.87820
2	0.90562	0.90325	0.89721	0.89719
3	0.89988	0.89924	0.89547	0.89579
4	0.90202	0.90054	0.88406	0.88671
5	0.90508	0.90493	0.90302	0.90409
6	0.89367	0.89019	0.89143	0.88966
7	0.89740	0.89833	0.88857	0.89169
8	0.90065	0.89727	0.88831	0.88611
9	0.89987	0.89758	0.88316	0.88285
Mean	0.90011	0.89815	0.88999	0.89012
Standard Deviation	0.00366	0.00450	0.00718	0.00748

Table S6 10-fold cross validation for our method and DeepSEA on H1-hESC dataset.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.90997	0.89778	0.89428	0.88395
1	0.91445	0.90644	0.91316	0.90454
2	0.91120	0.89931	0.90743	0.89670
3	0.91175	0.90423	0.89696	0.88851
4	0.91666	0.90623	0.90339	0.89508
5	0.91418	0.90379	0.90060	0.89198
6	0.90938	0.89838	0.91040	0.90055
7	0.91096	0.90206	0.91322	0.90612
8	0.90986	0.89956	0.91489	0.90567
9	0.91327	0.90102	0.91279	0.90305
Mean	0.91217	0.90188	0.90671	0.89761
Standard Deviation	0.00238	0.00318	0.00745	0.00769

Table S7 10-fold cross validation for our method and DeepSEA on HepG2 dataset.

Fold	Our method		DeepSEA	
	auROC	auPRC	auROC	auPRC
0	0.87610	0.86651	0.84508	0.83649
1	0.87225	0.86469	0.86917	0.86186
2	0.87442	0.86584	0.86780	0.85924
3	0.87090	0.86339	0.86009	0.85425
4	0.86569	0.85517	0.84669	0.83882
5	0.87018	0.85954	0.82295	0.80928
6	0.87640	0.86905	0.86234	0.85481
7	0.87224	0.86194	0.86297	0.85190
8	0.87574	0.86916	0.86737	0.86019
9	0.87244	0.86406	0.87255	0.86522
Mean	0.87264	0.86393	0.85770	0.84921
Standard Deviation	0.00328	0.00428	0.01524	0.01689

Table S8 Several runs with different random seeds of our model. We run the same model several times with different random seeds on the GM12878 dataset to show the stability of our model. We find that the mean auROC score for 10 runs of our model is 0.88070, with a very small standard deviation 0.00166. We conclude that our model has stability in performance.

run	auROC score	auPRC score
0	0.88136	0.87560
1	0.88292	0.87812
2	0.87898	0.87401
3	0.87954	0.87507
4	0.87781	0.87164
5	0.88301	0.87812
6	0.88004	0.87425
7	0.88092	0.87533
8	0.88157	0.87635
9	0.88087	0.87451
Mean	0.88070	0.87530
Standard Deviation	0.00166	0.00194

Table S9 *k*-mer pairs grouped by hamming distance. We compute the pairwise hamming distance between 4096 *k*-mers. Since there are always 6 characters in one *k*-mer, the hamming distance has only six possible values, i.e., [1, 2, 3, 4, 5, 6]. In each group, we look at the distribution of their corresponding *k*-mer vector distances, namely Euclidean distance and cosine distance, and for both we give the mean value with standard deviation shown in brackets.

Hamming distance	1	2	3	4	5	6
# pairs	36,864	276,480	1,105,920	2,488,320	2,985,984	1,492,992
Euclidean distance	4.5602 (0.6405)	5.6204 (0.6837)	6.0938 (0.5332)	6.3111 (0.5361)	6.4270 (0.5289)	6.4528 (0.5463)
Cosine distance	0.5272 (0.1466)	0.7964 (0.1730)	0.9292 (0.1418)	0.9958 (0.1435)	1.0322 (0.1412)	1.0408 (0.1464)

Table S10 *k*-mer pairs grouped by edit distance. We compute the pairwise edit distance between 4096 *k*-mers. Since there are always 6 characters in one *k*-mer, the edit distance has only six possible values, i.e., [1, 2, 3, 4, 5, 6]. The difference in edit distance and Hamming distance is that the former measure will align the strings before computing distance, so edit distance will be no larger than Hamming distance. In each group, we look at the distribution of their corresponding *k*-mer vector distances, namely Euclidean distance and cosine distance, and for both we give the mean value with standard deviation shown in brackets.

Edit distance	1	2	3	4	5	6
# pairs	36,864	355,494	1,602,378	3,272,994	2,560,482	558,348
Euclidean distance	4.5602 (0.6405)	5.6984 (0.6674)	6.1522 (0.5604)	6.3324 (0.5604)	6.4681 (0.4972)	6.5388 (0.5171)
Cosine distance	0.5272 (0.1466)	0.8157 (0.1695)	0.9434 (0.1414)	1.0006 (0.1485)	1.0470 (0.1320)	1.0822 (0.1352)

Table S11 Top 10 specific *k*-mers using *k*-mer rank as relative abundance. We here attempt to find the most specific *k*-mers corresponding to each cell line. First, we define the relative abundance of a *k*-mer as its rank according to frequency, which ranges from 0 to 4095. The smaller the rank is, the more enriched the *k*-mer is in that cell line. Second, we compute the relative abundance matrix for 4096 *k*-mers in 6 cell lines. Third, based on this matrix, we consider to measure the *k*-mer specificity score. The more heterogeneous the relative abundance is in six cell lines, the more specific the *k*-mer is. We simply compute the standard deviation of the six relative abundances for each *k*-mer as its specificity score. We pick out the top 10 *k*-mers with highest specificity scores, and give their ranks in each cell line and the final specificity score in the last column. we can find that 'atatat' and 'tatata' are more enriched in the first four cell lines than in H1-hESC and HepG2 cell lines, while the rest 8 *k*-mers, such as 'ccgcgg' etc., are more enriched in H1-hESC and HepG2 cell lines

k-mer	GM12878	K562	MCF-7	HeLa-S3	H1-hESC	HepG2	Specificity
atatat	1783	567	1341	619	3510	2006	994.5
ccgcgg	1453	2834	2375	2805	466	686	961.3
cgcggg	1535	2860	2473	2856	522	753	956.1
tatata	2084	755	1641	888	3614	2235	955.7
cccgcg	1579	2869	2470	2836	536	742	950.6
cgcggc	1512	2848	2384	2806	486	745	946.3
gccgcg	1490	2812	2358	2756	470	737	934.4
gcgcgg	1253	2701	2216	2637	378	643	929.8
ccgccg	967	2564	1942	2574	323	480	929.7
cggcgc	1755	2944	2558	2919	591	949	929.4

Table S12 Top 10 specific *k*-mers using normalized frequency as relative abundance (x1e-4). We again attempt to find the most specific *k*-mers corresponding to each cell line. Different from Table S11, we define the relative abundance of a *k*-mer as its normalized frequency, that is to say, every frequency number should be divided by the total number of *k*-mers in that cell line. Note that for simplicity, all values in this table are shown in 1e-4 unit.

k-mer	GM12878	K562	MCF-7	HeLa-S3	H1-hESC	HepG2	Specificity
aaaaaa	19.603	23.067	18.033	21.595	9.401	15.279	4.516
tttttt	19.608	22.470	17.736	21.482	9.657	15.031	4.330
cctccc	9.735	8.407	8.594	7.863	14.138	12.742	2.360
gggagg	9.648	8.424	8.611	7.964	14.183	12.534	2.320
ccctcc	7.518	6.454	6.607	6.062	12.177	10.438	2.289
ggaggg	7.581	6.501	6.616	6.210	12.259	10.415	2.277
ggcggg	5.808	3.700	4.288	3.894	9.246	7.626	2.063
cccgcc	5.728	3.648	4.236	3.821	9.164	7.570	2.057
aaaaat	7.431	8.967	7.399	9.515	3.517	5.430	2.046
attttt	7.393	9.015	7.432	9.459	3.476	5.503	2.045

Table S13 auROC scores for different embedding strategies. To prove the efficacy of *k*-mer embedding in our model, we propose anther two different embedding strategies. The average auROC scores for three strategies, namely '-init -train', '-init -notrain', '-noinit', are 0.8948, 0.8756, 0.8726, respectively. Our strategy '-init -train' is 0.0192 higher than '-init -notrain', showing that embedding vectors should be updated during the supervised training, since the initial *k*-mer embedding vectors are generated by unsupervised learning without seeing any information of data labels. Our strategy is 0.0222 higher than '-noinit', showing that the *k*-mer embedding vectors are actually a good initialization and informative features for classification. The strategy '-init -notrain' is slightly higher than '-noinit' also showing that the unsupervised *k*-mer embedding vectors are informative and helpful for classification.

	GM12878	K562	MCF-7	HeLa-S3	H1-hESC	HepG2
-init -train	0.8830	0.8809	0.9212	0.9016	0.9097	0.8722
-init -notrain	0.8638	0.8509	0.8998	0.8823	0.8986	0.8580
-noinit	0.8602	0.8545	0.8989	0.8806	0.8963	0.8452

Table S14 10-fold cross validation for three cases. To show the efficacy of the convolution stage and the BLSTM stage in our model, we propose another two variant deep learning architectures, one omitting the convolution stage and the other omitting the BLSTM stage. We carry out 10-fold cross validation on the GM12878 dataset for three cases. We find that the average auROC score for the three cases are 0.88146, 0.87926, 0.86797 respectively, showing that the full model with both convolution stage and BLSTM stage reaches the best performance.

Fold	full		nolstm		noconv	
	auROC	auPRC	auROC	auPRC	auROC	auPRC
0	0.88413	0.88222	0.87791	0.87479	0.86452	0.85542
1	0.87657	0.86888	0.88087	0.87474	0.86327	0.85528
2	0.88554	0.88233	0.87884	0.87454	0.88142	0.87530
3	0.87352	0.87033	0.87834	0.87644	0.85834	0.84981
4	0.87684	0.87284	0.87943	0.87598	0.87461	0.86836
5	0.88669	0.88299	0.88150	0.87654	0.86644	0.85849
6	0.88222	0.88032	0.87764	0.87463	0.86981	0.86303
7	0.88394	0.88214	0.87969	0.87695	0.86233	0.85515
8	0.88204	0.87998	0.87736	0.87381	0.87560	0.87029
9	0.88314	0.87782	0.88101	0.87588	0.86336	0.85397
Mean	0.88146	0.87798	0.87926	0.87543	0.86797	0.86051
Standard Deviation	0.00434	0.00534	0.00149	0.00105	0.00722	0.00834

Table S15 Wilcoxon tests results. To measure the significance of difference between three cases in Table S14, we perform one-sided Wilcoxon tests on the auROC and auPRC scores between our full network and the two variant networks. The *p*-values all prefer the alternative hypothesis that our full network has a higher auROC/auPRC score than the nolstm/noconv variant network, although the *p*-values of the full model and the nolstm model seems marginal.

p-values	auROC	auPRC
full vs. nolstm	0.07157	0.07157
full vs. noconv	0.0002057	0.0001299

Table S16 Comparison between BLSTM and LSTM. To compare the difference between BLSTM and LSTM, we modify the BLSTM layer into a standard LSTM layer in our network and retrain the model on the GM12878 dataset. We report the performance of the two networks, including auROC, auPRC, time for each epoch, number of epochs until convergence, and total training time. We find that, substituting LSTM for BLSTM, auROC only drops slightly while auPRC even shows a small increment, suggesting that BLSTM does not bring significant improvement compared to the standard LSTM network. For the running time, LSTM consumes less time for each epoch, but takes more epochs before convergence. In summary, we can choose either BLSTM or LSTM in a practical application.

	auROC	auPRC	Time (s/epoch)	# epochs	Total (h)
BLSTM	0.8830	0.8774	350	34	3.3
LSTM	0.8821	0.8778	334	40	3.7