PROTOTYPE MATCHING NETWORKS FOR LARGE-SCALE MULTI-LABEL CLASSIFICATION

Jack Lanchantin, Arshdeep Sekhon, Ritambhara Singh, & Yanjun Qi
Department of Computer Science
University of Virginia
Charlottesville,VA 22903, USA
{jjl5sw, as5cu, rs3zz, yanjun}@virginia.edu

ABSTRACT

One of the fundamental tasks in understanding genomics is the problem of predicting Transcription Factor Binding Sites (TFBSs). With more than hundreds of Transcription Factors (TFs) as labels, genomic-sequence based TFBS prediction is a challenging multi-label classification task. There are two major biological mechanisms for TF binding: (1) sequence-specific binding patterns on genomes known as "motifs" and (2) interactions among TFs known as co-binding effects. In this paper, we propose a novel deep architecture, the Prototype Matching Network (PMN) to mimic the TF binding mechanisms. Our PMN model automatically extracts prototypes for each TF through a novel prototype-matching loss. We use the notion of support set of prototypes and an LSTM to learn how TFs interact and bind to genomic sequences. On a TFBS dataset with 2.1 million genomic sequences, the PMN significantly outperforms baselines and validates our design choices empirically. Not only is the proposed architecture accurate, but also models the underlying biology.

1 INTRODUCTION

One of the fundamental tasks to understand genomes, and in turn diseases such as cancer, is the problem of predicting Transcription Factor Binding Sites (TFBSs) (Consortium et al., 2012). Transcription Factors (TFs) are proteins which bind (i.e., attach) to DNA and control whether a gene is expressed or not. TFs are known to bind to sequence-specific patterns on genomes, known as "motifs". If a TF binds in the absence of its motif, or it does not bind in the presence of its motif, then it is likely there are some external causes such as an interaction with another TF, known as co-binding effects (Wang et al., 2012). Thus, when designing a genomic-sequence based TFBS predictor, we should consider both: (1) how to automatically extract "motif"-like features and (2) how to model the co-binding patterns and consider such patterns in predicting TFBSs. In this paper, we address both proposing a novel deep-learning model: prototype matching network (PMN).

To extract motif-like features, we implement a CNN encoder, as commonly used in TFBS prediction. To model the dependencies among output labels, we introduce a combinationLSTM, which updates the encoder representation of the input conditioned on the output labels. The combinationLSTM models how the embedding of a test sample matches to a combination of relevant prototypes. Using multiple "hops", the combinationLSTM updates the embedding of the input sequence by searching for which TFs (prototypes) are more relevant in the label combination. Instead of explicitly modeling interactions among labels, we try to use the combinationLSTM to mimic the underlying biology. The combinationLSTM tries to learn prototype embedding and represent high-order label combinations through a weighted sum of prototype embedding. To our knowledge, this is the first deep learning architecture to model TF-TF interactions in an end-to-end model. On a reference TFBS dataset with 2.1 $million$ genomic sequences, PMN significantly outperforms the state-of-the-art TFBS prediction baselines.

2 PROTOTYPE MATCHING NETWORKS

Model Overview (shown in Figure 1) Given a DNA sequence x (composed of characters A,C,G,T) of length T, we want to classify x as a positive or negative binding site for each transcription factor $TF_1, ..., TF_\ell$ (i.e. multi-label binary classification). To do this, we seek to match x to a bank of ℓ learned TF prototype vectors, $\{p_1, ..., p_\ell\}$, where each prototype is loosely representative of the TF label. Additionally, since TFs may bind or not bind based on other TFs, we model the interactions among TFs before making a prediction.

Embedding The Sequence and Prototypes The input sequence $x \in \mathbb{R}^{4\times t}$ is encoded using a function f (3-layer CNN, which has shown to be sufficient for genomic feature extraction) to produce sequence embedding $\hat{x} \in \mathbb{R}^d$, $\hat{x} = f(x)$. The prototypes are produced by a multiplication of the identity matrix I and a learned lookup table matrix $W \in \mathbb{R}^{|TFs|\times d}$: $P = \mathrm{I}W$. We introduce a prototype matching loss $\mathcal{L}_p$, which forces a prototype to correspond to a specific TF (explained below).

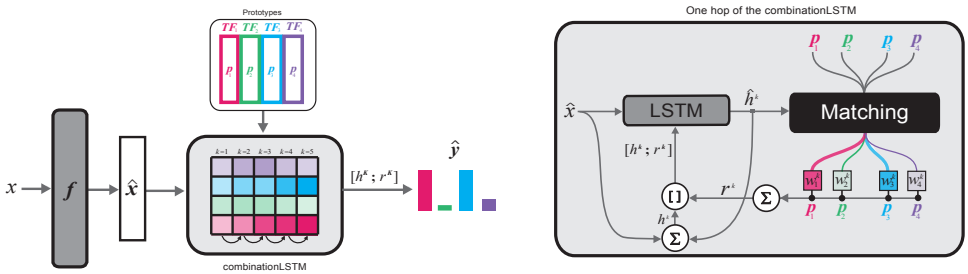

Figure 1: **Prototype Matching Network (PMN) Model**. On the left is an overview of the model. The input sequence x is encoded as $\hat{x}$ using f (3-layer CNN). $\hat{x}$ is then matched against the learned prototypes using the combinationLSTM for K "hops" so that it can update its output based on TF interactions for this input sequence. The final output $\hat{y}$ is based on a concatenation of the final updated sequence vector h^K from the LSTM, and final read vector r^K from the matching. On the right is a closer look at the combinationLSTM.

LSTM to learn Label Interactions and to Update the Sequence Embedding The main idea is that we want to update the sequence embedding $\hat{x}$ conditioned on matching against the prototypes. Since interactions among TFs influence binding, we cannot simply match the sequence to the prototypes. To obtain TF interactions, we use a combinationLSTM, similar to the attention LSTM in Vinyals et al. (2016). The combinationLSTM uses K "hops" to process the prototypes $p_1, p_2, ..., p_\ell$ by matching against an updated sequence embedding $\hat{h}^k$. The hops allow the combinationLSTM to update the output vector based on which TFs match simultaneously. At each hop, the LSTM accepts a constant $\hat{x}$, a concatenation of the previous LSTM hidden state h^{k-1} and read vector r^{k-1}, as well as the previous LSTM cell output c^{k-1}. h^0 and c^0 are initialized with zeros, and r^0 is initialized with the mean of all prototype vectors, $\frac{1}{|p|}\sum_i^{|p|} p_i$.

The output hidden state $\hat{h}^k$ is matched against each prototype using cosine similarity. Since this similarity is in the range [-1,1], we feed this output through a sigmoid function to produce the similarity score w_i^k at hop k in (0,1). The read vector r^k is updated by a weighted sum of the prototype vectors using the matching scores. At each hop, h^k is updated using the current LSTM output hidden state and sequence embedding.

$$\hat{h}^k, c^k = \text{LSTM}(\hat{x}, [h^{k-1}; r^{k-1}], c^{k-1}) \tag{1}$$

$$h^k = \hat{h}^k + \hat{x} \tag{2}$$

$$r^k = \sum_{i=1}^{|p|} w_i^k p_i \tag{3}$$

$$w_i^k = 1/\left(1 + e^{-\epsilon c(\hat{h}^k, p_i)}\right) \tag{4}$$

where $c(u, v)$ is cosine similarity, and ϵ is a hyperparameter of the sigmoid function that pushes the cosine similarity output closer to 0 or 1 (we use ϵ=20). In the TFBS task, eq. 5 is the important factor for modelling TF combinations. The read vector r^k can model multiple prototypes matching at once through a linear combination. Furthermore, the LSTM with K hops is needed because of the fact that a TF binding may influence other TFs in a sequential manner. For example, if TF_i matches to $\hat{h}^k$ in the first hop, r^k is then used to output $\hat{h}^{k+1}$ which can match to TF_j at the next hop. In this case, $\hat{h}^{k+1}$ is a joint representation of $\hat{x}$ and the current matched prototypes, represented by r^k. At each hop, the LSTM fine-tunes w^k in order to find TF binding combinations.

The final output $\hat{y} \in \mathbb{R}^{|TFs|}$ is computed from a concatenation of the final hidden state and read vectors $[h^K; r^K]$ after the Kth hop using a linear transform: $o = W([h^K; r^K])$. An element-wise sigmoid function is then applied to get a probability of binding for each TF: $\hat{y} = 1/(1 + e^{-o})$.

Classification and Prototype Matching Loss Functions To classify a sequence, we use a binary cross entropy loss between each label y_i for TF_i, and the corresponding TF_i output $\hat{y}_i$, which we call the classification loss, $\mathcal{L}_c$. We also introduce a prototype matching loss $\mathcal{L}_p$, which forces a prototype to correspond to a specific TF since prototypes are learned from random initializations. The prototype matching loss uses an L_2 loss between the true label y_i for TF_i and the final matching weight w_i^K between updated sequence h^K and prototype p_i. This loss forces a prototype to match to all of its positive binding sequences. The loss is computed from the final weights, w^K, after K hops, which allows the LSTM to attend to certain TFs at different hops before the final loss, modeling the co-binding of TFs. Hyperparameter λ controls the amount that each prototype is mapped to a specific TF. λ=0 corresponds to random prototypes. The final loss $\mathcal{L}$ is a summation of the classification loss and the prototype matching loss: $\mathcal{L} = -\mathcal{L}_c - \lambda\mathcal{L}_p$, where $\mathcal{L}_c = \sum_i^{|p|}(y_i \log\hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$, and

Table 1: TFBS Prediction Across 86 TFs in GM12878 Cell Line. We compare our PMN model to two CNN baseline models (single-label and multi-label). λ represents the weighting of the prototype loss in eq. 2. Our PMN model outperforms the CNN models across all 86 TFs.

Model	**auROC**			**auPR**			**Recall at 50% FDR**		
	Mean	Std.	% Increase over single	Mean	Std.	%Increase over single	Mean	Std.	%Increase over single
CNN (single-label)	0.820	0.072	-	0.263	0.123	-	0.224	0.198	-
CNN (multi-label)	0.831	0.055	1.37	0.257	0.113	-2.52	0.215	0.186	-4.00
PMN (λ=1), no LSTM	0.830	0.057	1.30	0.267	0.116	1.22	0.231	0.197	3.09
PMN (λ=1), softmax att	0.834	0.057	1.70	**0.272**	0.115	**3.36**	**0.243**	0.194	**8.48**
PMN (λ=0), sigmoid att	0.837	0.055	2.13	0.271	0.113	3.00	0.229	0.186	1.92
PMN (λ=0.5), sigmoid att	0.839	0.055	2.38	**0.272**	0.113	**3.36**	0.235	0.187	4.73
PMN (λ=1), sigmoid att	**0.840**	0.054	**2.45**	0.270	0.114	2.47	0.234	0.187	4.17

$\mathcal{L}_p = \sum_i^{|p|}(y_i - w_i^K)^2$. We trained our model using Adam (Kingma & Ba, 2014) for 40 epochs, and used dropout (Srivastava et al., 2014).

Connecting to Previous Studies Vinyals et al. (2015) and Vinyals et al. (2016) introduced general frameworks for processing input sets where they use an orderless LSTM, similar to the one we use. In our model, we use a set of learned prototypes as proxies to model the dependencies among output labels before making a prediction. Snell et al. (2017) introduces prototypical networks for zero and one-shot learning, which assumes that the data points belonging to a particular class cluster around a single prototype. Krotov & Hopfield (2016) show that pattern recognition is likely a combination of both feature-matching (i.e. LSTM or CNN encoders) and prototype-matching (i.e. comparing against learned prototypes). Our method is motivated by the prototype-matching theory, where instead of searching for exact features to match against, the model tests an unseen sample against a set of prototypes using a defined similarity metric to make a classification. Our method is also similar to Munkhdalai & Yu (2016) for QA, which uses a vanilla LSTM to update a query based on the document to select an answer from the document.

3 Experiments and Results

Dataset: We constructed our dataset from ChIP-seq experiments in the ENCODE project database (Consortium et al., 2012). We divided the entire human genome up into 200-length sequences, using a sliding window of 50. We then extracted the 200-length windows surrounding the peak locations for 86 transcription factors in the human lymphoblastoid cell line (GM12878). Our training, validation, and test sets contain 1.44M, 330K, 300K sequences, respectively. Each sample has an average of $\sim$ 5 positive labels (i.e. TFs binding).

Model Variations: To test the PMN model on our TFBS dataset, we constructed 4 model variations:
1. CNN: Similar to (Zhou et al.; Lanchantin et al., 2016), we use a baseline 3-layer CNN model. We use $\{512,256,128\}$ kernels of widths $\{9,5,3\}$ at the 3 layers, respectively. The output of the CNN is maxpooled across the length, resulting in a final output vector of size 128. This architecture is used as the encoder module for all variations. We also implement a baseline single-task CNN which assumes no interactions among TFs.
2. PMN no LSTM: We use eq. 5-9, except that we replace $\hat{h}^k$ in eq. 9 with $\hat{x}$ since there is no LSTM. The output (eq. 11) is then a concatenation of r and $\hat{x}$. We still use the full prototype loss ($\lambda = 1$).
3. PMN, sigmoid att: The full PMN model utilizes the CNN plus the combinationLSTM module in eq. 6 over K hops (We use $K = 5$ hops since each sample has on average 5 positive labels). We tested 3 variations of the prototype loss ($\lambda = 0, \lambda = 0.5, \lambda = 1$), where $\lambda = 0$ represents no prototype loss, or random prototypes.
4. PMN, softmax att: Since softmax attention is typically used in attention models, we explored using a softmax function to replace eq. 6 from $k = 0$ until $k = K - 1$, and then an element-wise sigmoid function (eq. 6) for the final multi-label classification output.

Large-Scale TFBS Classification Results Table 1 shows the results of our models across the 86 TF labels. The joint CNN (multi-label) model outperformed the single label CNN models in auROC and auPR. The main advantage of the joint model is that it is faster than an individual model for each TF. The joint model's improvement over the single-task models was not significant (p-value < 0.05) based on a one-tailed pairwise t-test. This is presumably because the joint model finds motifs similar among all motifs, but it doesn't model interactions among TF labels. The PMN model outperformed both baseline CNN models in all 3 metrics. In addition, the improvement of the PMN over both CNN models was significant using a one-tailed pairwise t-test. We hypothesize that the combinationLSTM module accurately models co-binding better, leading to an increase in performance.

asdas

REFERENCES

ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems*, pp. 1172–1180, 2016.

Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. *arXiv preprint arXiv:1608.03644*, 2016.

Tsendsuren Munkhdalai and Hong Yu. Reasoning with memory augmented neural networks for language comprehension. *arXiv preprint arXiv:1610.06454*, 2016.

Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1): 1929–1958, 2014.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.

Jie Wang, Jiali Zhuang, Sowmya Iyer, XinYing Lin, Troy W Whitfield, Melissa C Greven, Brian G Pierce, Xianjun Dong, Anshul Kundaje, Yong Cheng, et al. Sequence features and chromatin structure around the genomic regions bound by 119 human transcription factors. *Genome research*, 22(9):1798–1812, 2012.

Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Simple baseline for visual question answering. URL `https://arxiv.org/abs/1512.02167`.