Deep GDashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks

Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
{jjl5sw,rs3zz,bw4mw,yq2h}@virginia.edu

Abstract

Deep neural network (DNN) models have recently obtained state-of-the-art prediction accuracy for the transcription factor binding (TFBS) site classification task. However, it remains unclear how these approaches identify meaningful DNA sequence signals and give insights as to why TFs bind to certain locations. In this paper, we propose a toolkit called the Deep Genomic Dashboard (Deep GDashboard) which provides a suite of visualization strategies to extract motifs, or sequence patterns from deep neural network models for TFBS classification. We demonstrate how to visualize and understand three important DNN models: convolutional, recurrent, and convolutional-recurrent networks. Our first visualization method is finding a test sequence's saliency map which uses first-order derivatives to describe the importance of each nucleotide in making the final prediction. Second, considering recurrent models make predictions in a temporal manner (from one end of a TFBS sequence to the other), we introduce temporal output values, indicating the prediction score of a model over time for a sequential input. Lastly, a class-specific visualization strategy finds the optimal input sequence for a given TFBS positive class via stochastic gradient optimization. Our experimental results indicate that a convolutional-recurrent architecture performs the best among the three architectures. The visualization techniques indicate that CNN-RNN makes predictions by modeling both motifs as well as dependencies among them.

1 Introduction

Understanding genetic sequences is one of the fundamental tasks of health advancements due to the high correlation of genes with diseases and drugs. An important problem within genetic sequence understanding is related to transcription factors (TFs), which are regulatory proteins that bind to DNA. Each different TF binds to specific transcription factor binding sites (TFBSs) on the genome to regulate cell machinery. Given an input DNA sequence, classifying whether or not there is a binding site for a particular TF is a core task of bioinformatics[23].

In recent years, there has been a revolution in deep learning models which have lead to groundbreaking results in many fields such as computer vision[13], natural language processing[24], and computational biology [2, 19, 26, 11, 14]. Using a convolutional model, Alipanahi et al.[2] was the first to achieve state-of-the-art results with a neural network on the TFBS classification task. Lanchantin et al.[14] showed that a deeper CNN model was able to outperform Alipanahi et al.[2]. Zhou et al.[26] employed a similar convolutional model to Alipanahi et al.[2], but focused on predicting noncoding genomic variations from chromatin-profiling data, where TFBS prediction was an intermediate step to the end goal. Later, Quang and Xie [19] showed that a convolutional-recurrent network combination outperformed Zhou et al.[26] on the same task.

Although DNN models have produced high accuracy predictions, the inner-workings and individual predictions of the models are hard to interpret due to black box nature of DNNs which automatically learn useful features. Aiming to open up the black box, in this paper we present the "Deep Genomic Dashboard[1]" (Deep GDashboard) to find key

---

[1]Dashboard normally refers to a user interface that gives a current summary, usually in graphic, easy-to-read form, of key information relating to performance[1].

information to interpret how state-of-the-art deep models make predictions on a genomic sequence classification task. We explore three visualization methods for interpretation:

1. Measuring nucleotide importance with **Saliency Maps**.
2. Measuring critical sequence positions for the classifier output using **Temporal Output Scores**.
3. Generating class-specific motif patterns with **Class Optimization**.

To visualize and understand how different models perform on the TFBS task, we select three deep learning architectures that have shown impressive results on benchmark tasks. Section 2 introduces the three different DNN structures that we use to evaluate TFBS prediction: convolutional neural networks (CNNs), recurrent neural networks (RNNs), and convolutional-recurrent neural networks (CNN-RNNs). In section 3, we provide detailed descriptions of the three visualization strategies. Section 4 provides experimental results for understanding and visualizing the three DNN architectures. We find that the CNN-RNN outperforms the others on a large scale benchmark TFBS classification dataset. From the visualizations, we observe that CNN-RNN tends to focus its predictions on the traditional motifs, as well as modeling long range dependencies among motifs.

2 Deep Neural Models for TFBS Classification

TFBS Classification. Chromatin immunoprecipitation (ChIP-seq) technologies and databases such as ENCODE [5] have made binding site sequences available for hundreds of different TFs. Despite these advancements, there are two major drawbacks: (1) ChIP-seq experiments are slow and expensive, (2) although ChIP-seq experiments can find the binding site locations, they cannot find patterns that are common across all of the positive binding sites which can give insight as to why TFs bind to those locations. Thus, there is a need for large scale computational methods that can not only make accurate binding site classifications, but also identify and understand patterns that represent the positive binding sites.

In order to computationally predict TFBSs, researchers initially used consensus sequences and position weight matrices to match against a test sequence [23]. Simple neural network classifiers were then proposed to differentiate positive and negative binding sites, but did not show significant improvements over the weight matrix matching methods [9]. Later, SVMs techniques outperformed the generative methods by using k-mer features [6, 20], but the string kernel based SVM systems are limited by expensive computational cost proportional to the number of training and testing sequences. Most recently, convolutional neural network models have shown state-of-the-art results on the TFBS task and are scalable to a large number of genomic sequences [2].

Deep Neural Networks for TFBSs. Inspired by their success across different fields, we explore variations of two popular deep learning architectures for TFBS prediction: convolutional neural networks (CNNs), and recurrent neural networks (RNNs). CNNs have dominated the field of computer vision in recent years, obtaining state-of-the-art results in many tasks due to their ability to automatically extract translation-invariant features. On the other hand, RNNs have emerged as one of the most powerful models for sequential data tasks such as natural language processing due to their ability to learn long range dependencies. More specifically, on the TFBS prediction task, we explore three distinct architectures, a (1) CNN, a (2) RNN, and (3) combination of the two, CNN-RNN. An overview of each model can be seen in Fig. 1.

End-to-end Deep Framework. While the body of the deep architectures differ, each implemented model follows a similar end-to-end framework which we use to easily compare and contrast results. We use the raw nucleotide characters as inputs, each of which are converted into a one-hot encoding (a binary vector with the matching the character entry as 1 and rest as 0s). This encoding matrix is used as the input to a convolutional, recurrent, or convolutional-recurrent module that each outputs a vector of fixed dimension. The output of the framework is a probability indicating whether an input is a positive or a negative binding site (binary classification task). This deep framework uses a softmax function as the last layer that learns the mapping from a hidden space to the output class label space $C \in [+1, -1]$. The parameters of the network are trained end-to-end by minimizing the negative log-likelihood over the training set. The minimization of the loss function is obtained via the stochastic gradient algorithm Adam[12], with a mini-batch size of 256 sequences. We use dropout [22] as a regularization method for each model.

2.1 Convolutional Neural Network (CNN)

In genomic sequences, it is believed that regulatory mechanisms such as transcription factor binding are influenced by local sequential patterns known as "motifs". Motifs can be viewed as the temporal equivalent of local spatial patterns
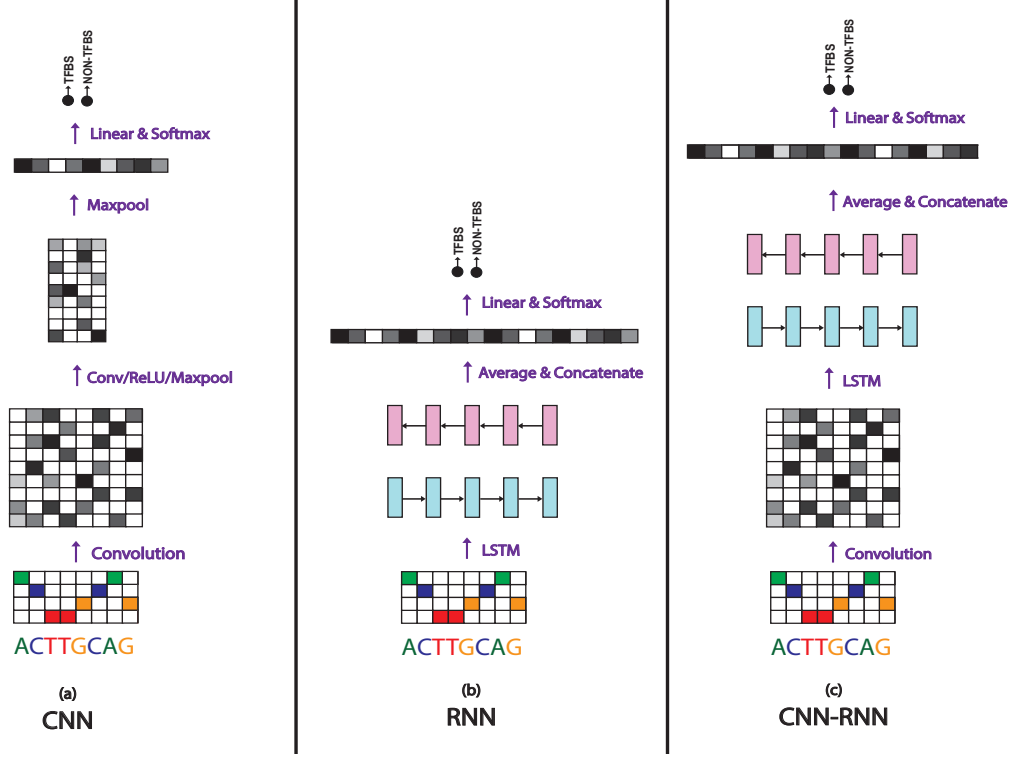
Figure 1: **Model Architectures.** Each model has the same input (one-hot encoded matrix of the raw nucleotide inputs), and the same output (softmax classifier to make a binary prediction). The architectures differ by the middle "module", which are **(a)** Convolutional, **(b)** Recurrent, and **(c)** Convolutional-Recurrent.

in images such as eyes on a face, which is what CNNs are able to automatically learn and achieve state-of-the art results on vision tasks. As a result, a temporal convolutional neural network is a fitting model to automatically extract these motifs. A temporal convolution with kernel size k takes an input of sequential data matrix $\mathbf{X}$ of size $T \times n_{in}$ with length T, and input layer size n_{in}. This convolution outputs a matrix $\mathbf{Y}$ of size $T \times n_{out}$, where n_{out} is the output layer size. Specifically, $convolution(\mathbf{X}) = \mathbf{Z}$, where

$$\mathbf{z}_{t,i} = \sigma(\mathbf{B}_i + \sum_{j=1}^{n_{in}} \sum_{z=1}^{k} \mathbf{W}_{i,j,z}\mathbf{x}_{t+z-1,j}), \tag{1}$$

where $\mathbf{W}$ and $\mathbf{B}$ are the trainable parameters of the convolution kernel, and σ is a function enforcing element-wise nonlinearity. We use rectified linear units (ReLU) as the nonlinearity:

$$\text{ReLU}(x) = \max(0, x). \tag{2}$$

After the convolution and nonlinearity, CNNs typically use maxpooling, which is a dimension reduction technique to provide translation invariance and to extract higher level features from a wider range of the input sequence. Temporal maxpooling on a matrix $\mathbf{Z}$ with a pooling size of m results in output matrix $\mathbf{Z}$. In summary, $maxpool(\mathbf{Z}) = \mathbf{Y}$, where

$$\mathbf{y}_{t,i} = \max_{j=1}^{m} \mathbf{z}_{m(t-1)+j,i} \tag{3}$$

Our CNN implementation involves a progression of convolution, non-linearity, and max-pooling. This is represented as one convolutional layer in the deep network, and we test up to 4 layer deep CNNs. The final layer involves a maxpool across the entire temporal domain so that we have a fixed-size vector which can be fed into a softmax classifier layer.

2.2 Recurrent Neural Network (RNN)

Designed to handle sequential data, Recurrent neural networks (RNNs) have become the main neural model for tasks such as natural language processing. The key advantage of RNNs over CNNs is that they are able to find long range patterns in the data which are highly dependent on the ordering of the sequence for the prediction task.

Given an input matrix $\mathbf{X}$ of size $T \times n_{in}$, an RNN produces matrix $\mathbf{H}$ of size $T \times d$. At each time step t, an RNN takes an input column vector $\mathbf{x_t} \in \mathbb{R}^{n_{in}}$ and a hidden state vector $\mathbf{h_{t-1}} \in \mathbb{R}^{n_{out}}$ and produces the next hidden state $\mathbf{h_t}$ by applying the following recursive operation:

$$h_t = \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1} + \mathbf{b}), \qquad (4)$$

where $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are the trainable parameters of the model, and σ is an element-wise nonlinearity. Due to their recursive nature, RNNs can model the full conditional distribution of any sequential data and find dependencies over time, where each position in a sequence is a time step on an imaginary time coordinate running in a certain direction. To handle the "vanishing gradients" problem of training basic RNNs on long sequences, Hochreiter and Schmidhuber [8] proposed an RNN variant called the Long Short-term Memory (LSTM) network (for simplicity, we refer to LSTMs as RNNs in this paper), which can handle long term dependencies by using gating functions. These gates can control when information is written to, read from, and forgotten. Specifically, LSTM "cells" take inputs $\mathbf{x}_t, \mathbf{h}_{t-1}$, and $\mathbf{c}_{t-1}$, and produce $\mathbf{h}_t$, and $\mathbf{c}_t$:

$$\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1} + \mathbf{b}^i)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1} + \mathbf{b}^f)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1} + \mathbf{b}^o)$$
$$\mathbf{g}_t = tanh(\mathbf{W}^g\mathbf{x}_t + \mathbf{U}^g\mathbf{h}_{t-1} + \mathbf{b}^g)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot tanh(\mathbf{c}_t)$$

where $\sigma(\cdot)$ and $tanh(\cdot)$ are element-wise sigmoid and hyperbolic tangent functions. $\odot$ represents an element-wise multiplication. $\mathbf{i}_t, \mathbf{f}_t$, and $\mathbf{o}_t$ are referred to as the input, forget, and output gates, respectively.

RNNs produce an output vector $\mathbf{h}_t$ at each time step of the input sequence. In order to use them on a classification task, we take the mean of all vectors at each time step, and use the mean vector as the input to the softmax layer. The mean vector is in $\mathbb{R}^d$, where d is the LSTM embedding size.

Because there is no innate direction in genomic sequences, we use a bi-directional LSTM as our RNN model. In the bi-directional LSTM ,the input sequence gets fed through two LSTM networks, one in each direction, and then the outputs vectors of each direction get concatenated together in the temporal direction and fed through a linear classifier.

2.3   Convolutional-Recurrent Network (CNN-RNN)

Since convolutional networks are designed to extract motifs, and recurrent networks are designed to extract temporal features, a combination of the two should be able to find temporal patterns between the motifs. Given an input matrix $\mathbf{X} \in \mathbb{R}^{T \times n_{in}}$, the output of the CNN is $\mathbf{Z} \in \mathbb{R}^{T \times n_{out}}$. Each column vector of $\mathbf{Z}$ gets fed into the RNN one at a time in the same way that the one-hot encoded raw vectors get input to the regular RNN model. The resulting output of the RNN is $\mathbf{H} \in \mathbb{R}^{T \times d}$, where d is then averaged across the temporal domain (in the same way as the regular RNN), and fed to a soft-max classifier layer.

3   Deep GDashboard: Visualizing and Understanding Deep Models

While making accurate predictions is important in biomedical tasks, it is equally important to understand the model. Accurate, but uninterpretable models are often very slow to emerge in practice due to the inability to understand their predictions, making domain experts reluctant to use them. Contrary to previous DNN works which focus on high accuracy classifications for genomic sequence tasks, we aim to obtain a better understanding of why certain models work better than others, and investigate how they make their predictions.

The visualization strategies we explore in the Deep GDashboard are inspired by recent work attempting to understand the deep models in computer vision and natural language processing. For example, given a deep network trained to classify objects in an image, trying to find patterns in the network which correspond to how humans would classify such image [21, 25]. Similarly, given a neural network trained to generate natural language, trying to find interpretable cells of the network which computationally follow known language semantics [10]. The main difference in our work is that instead of trying to understand the neural models given human understanding of such human perception tasks, the Deep GDashboard attempts to uncover critical signals in DNA sequence given the deep models. The Deep GDashboard allows us visualize and understand the DNN models in 3 different ways: Saliency Maps, Temporal Output Scores, and Class Optimizations.

3.1 Saliency Maps

Given a certain DNA sequence, it is useful to interpret which parts of the sequence are most influential to its classification score given a DNN of interest. Similar to the methods used in computer vision by Simonyan et al.[21] and Baehrens et al.[4], we seek to visualize the influence of each position (i.e. nucleotide) in the sequence. Given a sequence X_0, and class $c \in C$, a DNN model provides a score function $S_c(X_0)$. We rank the nucleotides of X_0 based on their influence on the score $S_c(X_0)$. Since $S_c(X)$ is a highly non-linear function of X with deep neural nets, it is hard to directly see the influence of each nucleotide of X on S_c. Mathematically, around the point X_0, $S_c(X)$ can be approximated by a linear function by computing the first-order Taylor expansion:

$$S_c(X) \approx w^T X + b \tag{5}$$

where w is the derivative of S_c with respect to the sequence variable X at the point X_0:

$$w = \left. \frac{\partial S_c}{\partial X} \right|_{X_0} \tag{6}$$

This derivative is simply one step of backpropagation in the DNN model, therefore it is easy to compute. We do a pointwise multiplication of the saliency map with the one-hot encoded sequence to get the derivative values for the actual nucleotide characters of the sequence (A,T,C, or G) so we can see the influence of the character at each position on the output score. Finally, we take the element-wise magnitude of the resulting derivative vector to visualize how important each character is regardless of derivative direction. We call the resulting vector a "saliency map[21]" because it tells us which nucleotides need to be changed the least in order to affect the class score the most.

3.2 Temporal Output Scores

Since genomic sequences are sequential, it can be insightful to visualize the output scores at each time step (position) of a sequence, which we call the temporal output scores. Here we assume an imaginary time direction running from left to right on a given sequence, so each position in the sequence is a time step in such an imagined time coordinate. In other words, we check the RNN's prediction scores when we vary the input of the RNN. The input series is constructed by using subsequences of an input X running along the imaginary time coordinate, where the subsequences start from just the first nucleotide (position), and ends with the entire sequence X. This way we can see exactly where in the sequence the recurrent model changes its decision from negative to positive, or vice versa. Since our recurrent models are bi-directional, we also use the same technique on the reverse sequence. CNNs process the entire sequence at once, thus we can't view its output as a temporal sequence, so we use this visualization on just the RNN and CNN-RNN.

3.3 Class Optimization

The previous two visualization methods listed are representative of a specific testing sample (i.e. sequence-specific). Now we introduce an approach to extract a *class-specific* visualization for a DNN model, where attempt to find the best sequence which maximizes the probability of a positive TFBS. Formally, we optimize the following equation where $S_+(X)$ is the probability (or score) of an input sequence X being a positive TFBS class output computed by the softmax equation of our trained DNN model for a specific TF:

$$\arg \max_X S_+(X) + \lambda \|X\|_2^2 \tag{7}$$

where λ is the regularization parameter. We find a locally optimal X through stochastic gradient descent, where the optimization is with respect to the input sequence. In this optimization, the model weights remain unchanged. This is similar to the methods used in Simonyan et al.[21] to optimize toward a specific image class. This visualization method depicts the notion of a positive TFBS class for a particular TF and is not specific to any test sequence.

3.4 Connecting to Previous Studies

Neural networks have produced state-of-the-art results on several important benchmark tasks related to genomic sequence classification [2, 26], making them a good candidate to use. However, *why* these approaches work well has been poorly understood. Recent works have attempted to uncover the properties of these models, in which most of the work has been done on understanding image classifications using convolutional neural networks. Zeiler et al.[25] used a "deconvolution" approach to map hidden layer representations back to the input space for a specific example, showing the features of the image which were important for classification. Simonyan et al.[21] explored a similar approach by using a first-order Taylor expansion to linearly approximate the network and find the input

Model	Conv. Layers	Conv. Size (n_{out})	Conv. Kernel Sizes (k)	Conv. Pool Size (m)	LSTM Layers	LSTM Size (d)
Small RNN	N/A	N/A	N/A	N/A	1	32
Large RNN	N/A	N/A	N/A	N/A	2	32
Small CNN	2	64	9,5	2	N/A	N/A
Medium CNN	3	64	9,5,3	2	N/A	N/A
Large CNN	4	64	9,5,3,3	2	N/A	N/A
Small CNN-RNN	1	128	9	0	1	32
Large CNN-RNN	2	128	9,5	2	1	32

Table 1: Variations of DNN Model Hyperparameters

features most relevant, while also optimizing image classes. Many similar techniques later followed to understand convolutional models [17, 3]. Most importantly, researchers have found that CNNs are able to extract layers of translational-invariant feature maps, which may indicate why CNNs have been successfully used in genomic sequence predictions which are triggered by motifs.

On text-based tasks, there have been fewer visualization studies for DNNs. Karpathy et al.[10] explored the interpretability of RNNs on a task of language modeling and found that there exist interpretable "cells" which are able to focus on certain language structure such as quotes. Li et al.[15] visualized how RNNs achieve compositionality in natural language for sentiment analysis by visualizing RNN embedding vectors as well as measuring the influence of input words on classification. Both studies have shown examples that can be validated by our understanding of natural language linguistics. We are instead interested in understanding DNA "linguistics" by our understanding of neural networks (the opposite direction of Karpathy et al.[10] and Li et al.[15]).

For TFBS prediction, Alipanahi et al.[2] was the first to implement a visualization method on a DNN model. They visualize their convolutional model by extracting motifs based on the input subsequence corresponding to the strongest activation location for each convolutional filter. Since they only have one convolutional layer, it is trivial to map the activations back. However, this method does not work as well when working with deeper, or multiple layers. We attempted this technique on our models and found that our approach using saliency maps outperforms it in finding motif patterns (details in section 4). Quang and Xie et al.[19] use the same visualization method on their convolutional-recurrent model for noncoding variation prediction.

4 Experiments and Results

4.1 Experimental Setup

Dataset. In order to evaluate our visualization approaches, we train and test DNN models on the 108 K562 cell ENCODE ChIP-Seq TF datasets used in Alipanahi et al.[2]. Each TF dataset has an average of 30,819 training sequences (with an even positive/negative split), and each sequence consists of 101 DNA-base characters (A,C,G,T). Every dataset has 1,000 testing sequences (with an even positive/negative split). Positive sequences are extracted from the hg19 genome centered at the reported ChIP-Seq peak. Negative sequences are generated by dinucleotide-preserving shuffle of the positive sequences. Due to the separate train/test data for each TF, we train a separate model for each individual TF dataset.

Variations of DNN Models. We implement several variations of each DNN architecture by varying the hyperparameters. Table 1 shows the different hyperparameter variations tested in each architecture.

Accuracy Evaluation. To compare prediction performance, we use area under ROC curve (AUC), which measures how well a binary classification model separates the two classes on a scale from 0 to 1, with 1 indicating a perfect separation. To check the statistical significance of such performance comparison, we implement a pairwise t-test and report two-tailed p-values.

Visualization Evaluation. To formally compare visualization methods, we evaluate each DNN's capability to find motifs, or consensus subsequences that significantly match to JASPAR[18] motif for each TF using the Tomtom[7] tool. Tomtom searches a query motif against a given motif database (and their reverse complements), and returns significant matches ranked by p-value indicating motif-motif similarity. Motif extraction for each of the models is implemented as follows. (1) From each positive test sequence (thus, 500 total for each TF dataset) we extract the saliency maps derived motif by selecting the contiguous length-9 subsequence that achieves the highest sum

Model	Mean AUC	Median AUC	STDEV
MEME-ChIP [16]	0.834	0.868	0.127
DeepBind [2] (CNN)	0.903	0.931	0.091
Small RNN	0.876	0.905	0.116
Large RNN	0.808	0.860	0.175
Small CNN	0.896	0.918	0.098
Med CNN	0.902	0.922	0.085
Large CNN	0.880	0.890	0.093
Small CNN-RNN	**0.925**	**0.947**	**0.073**
Large CNN-RNN	0.918	0.944	0.081

Table 2: Mean AUC scores on the TFBS classification task

Model Comparison[2]	p-value
RNN vs MEME	5.15E-05
CNN vs MEME	1.87E-19
CNN-RNN vs MEME	4.84E-24
CNN vs RNN	5.08E-04
CNN-RNN vs RNN	7.99E-10
CNN-RNN vs CNN	4.79E-22

Table 3: AUC pair-wise t-test

of contiguous length-9 saliency map values. (2) For each positive test sequence, when extracting motifs using the temporal output values, we select the length-9 subsequences that obtains the strongest score change from negative to positive output score. (3) Lastly, we compare each of the class-optimized sequences using the class optimization method against the JASPAR motifs.

Baselines. We use the "MEME-ChIP [16] sum" results from Alipanahi et al.[2] as one prediction performance baseline. This paper [2] applied MEME-ChIP to the top 500 positive training sequences, derives five PWMs, and scores test sequences using the sum of scores using all five PWMs. We also compare against the CNN model proposed in Alipanahi et al.[2]. For the visualization (motif) evaluation, we compare against the "conv activation" method used in Alipanahi et al.[2] and Quang and Xie [19], where we map the strongest first layer convolution filter activation back to the input sequence to find the most influential length-9 subsequence.

4.2 Prediction Performance of DNN Models

Table 2 shows the average AUC scores for each of the tested models (summarized in Table 1). As expected, the CNN model outperforms the standard RNN model. This matches our hypothesis that positive binding sites are mainly triggered by local patterns or "motifs" that CNNs can easily find. Interestingly, the CNN-RNN achieves the best performance among three deep architectures we tried. We hypothesize that this signifies besides modeling motifs, it is also important to model sequential interactions among motifs. In the CNN-RNN combination, CNN acts like a "motif finder" and the RNN finds dependencies among motifs. To check the statistical significance of such comparisons, we apply a pairwise t-test using the AUC scores for each TF and report the two tailed p-values in Table 3. All deep models are significantly better than the MEME baseline. The CNN is significantly better than the RNN and the CNN-RNN is significantly better than the CNN.

4.3 Deep GDashboard

Table 4 summarizes the motif matching results comparing visualization-derived motifs against known motifs in the JASPAR database [18]. We are limited to a comparison of 57 out of our 108 TF datasets by the TFs which JASPAR has motifs for. Table 4 compares four visualization approaches: Saliency Map, Convolution Activation, Temporal Output Scores and Class Optimizations. The first three techniques are test-sequence specific, therefore, we report the average number of motif matches out of 500 positive sequences (being averaged across 57 TF datasets). The last one is for a particular TFBS positive class. We can see from Table 4, across multiple visualization techniques, the CNN finds motifs the best, followed by the CNN-RNN and the RNN. Since CNNs perform worse than CNN-RNNs, we hypothesize that it is not only the motifs that are critical for a TF to bind, but also the temporal interactions among them. This analysis clearly shows that visualizing the DNN classifications can lead to a better understanding of DNNs for TFBSs.

Figure 2 shows examples of the Deep GDashboard for three different TFs and sequences. We apply the visualization on the best performing models of each of the three DNN architectures. Each Deep GDashboard snapshot is for a specific TF and contains (1) JASPAR motifs for that TF, (2) the positive TFBS class-optimized sequence for each architecture (for the given TF of interest), (3) the test sequence of interest, where the JASPAR motifs in the test sequences are highlighted using a pink box, (4) the saliency map from each DNN model on the given sequence, and (5) forward and backward temporal output values from the recurrent architectures. In the saliency maps, the more red a position is, the more influential it is for the prediction. In the temporal outputs, blue indicates a negative TFBS prediction while red indicates positive. The saliency map and temporal output visualizations are on the same positive test sequence (as shown twice). The numbers next to the model names in the saliency map section indicate the score outputs of that DNN model on the specified test sequence.

Figure 2: **Deep GDashboard**. Dashboard examples for GATA1, MAFK, and NFYB positive TFBS Sequences. The top section of the dashboard contains the Class Optimization (which does not pertain to a specific test sequence, but rather the class in general). The middle section contains the Saliency Maps for a specific positive test sequence, and the bottom section contains the temporal Output Values for the same positive test sequence used in the saliency map. The very top contains known JASPAR motifs, which are highlighted by pink boxes in the test sequences if they contain motifs.

	Saliency Map (out of 500)	Conv. Activations[2, 19] (out of 500)	Temporal Output (out of 500)	Class Optimization (out of 57)
CNN	243.9	173.4	N/A	19
RNN	168.1	N/A	53.5	11
CNN-RNN	138.6	74.2	113.2	13

Table 4: JASPAR motif matches against Deep GDashboard and baseline motif finding methods using Tomtom

Saliency Maps (middle section of dashboard). By visual inspection, we can see from the saliency maps that CNNs tend to focus on short contiguous subsequences when predicting positive bindings. In other words, CNNs clearly model "motifs" that are the most influential for prediction. The saliency maps of RNNs tend to be spread out more across the entire sequence, indicating that they focus on all nucleotides together, and infer relationships among them. The CNN-RNNs have strong saliency map values around motifs, but we can also see that there are other nucleotides further away from the motifs that are influential for the model's prediction. For example, the CNN-RNN model is 99% confident in its GATA1 TFBS prediction, but the prediction is also influenced by nucleotides outside the motif. In the MAFK saliency maps, we can see that the CNN-RNN and RNN focus on a very wide range of nucleotides to make their predictions, and the RNN doesn't even focus on the known JASPAR motif to make its high confidence prediction. The NFYB sequence shows an example where the CNN and RNN perform poorly, but the CNN-RNN makes the correct prediction. In this test sequence, there are no clear motifs, which is presumably why the CNN makes an incorrect prediction. We assume that although the CNN focuses on the same two ranges as the CNN-RNN does, it can't model temporal interactions among the two, which is necessary for predicting the binding.

Output Values (bottom section of dashboard). For most of the sequences that we tested, motifs from the temporal output values are located precisely at those positions that trigger the model to switch from a negative TFBS prediction to positive. For some other sequences, however, the nucleotides flanking the motif are also found to be influential in making the switch from negative to positive at the correct temporal position. We did not observe clear differences between the forward and backward temporal output patterns.

Class Optimization (top section of dashboard). Class optimization on the CNN model generates concise representations which often resemble the known motifs for that particular TF. For the recurrent models, the TFBS positive optimizations are less clear, though some aspects stand out (like "AT" followed by "TC" in the GATA1 TF for the CNN-RNN). We notice that for certain DNN models, their class optimized sequences optimize the reverse complement motif (e.g. NFYB CNN optimization). The class optimizations can be useful for getting a general idea of what triggers a positive TFBS for a certain TF.

5 Conclusion

Deep neural networks have shown to be the most accurate models in genomic sequence based TFBS classification tasks. However, since they are hard to interpret, their adaptation in practice is slow. In this work, we perform an empirical evaluation of 3 different DNN architectures on TFBS prediction, and introduce three visualization methods to shed light on how these models work. We hope that this work will invoke further studies on visualizing and understanding DNN based genomic sequences analysis.

References

[1] Dashboard definiton. `http://www.dictionary.com/browse/dashboard`. Accessed: 2016-07-20.

[2] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. Nature Publishing Group, 2015.

[3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. volume 10, page e0130140, 2015.

[4] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. volume 11, pages 1803–1831, 2010.

[5] ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. volume 489, pages 57–74. Nature Publishing Group, 2012.

[6] Mahmoud Ghandi, Dongwon Lee, Morteza Mohammad-Noori, and Michael A Beer. Enhanced regulatory sequence prediction using gapped k-mer features. 2014.

[7] Shobhit Gupta, John A Stamatoyannopoulos, Timothy L Bailey, and William S Noble. Quantifying similarity between motifs. volume 8, page R24. BioMed Central Ltd, 2007.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. volume 9, pages 1735–1780. MIT Press, 1997.

[9] Paul B Horton and Minoru Kanehisa. An assessment of neural network and statistical approaches for prediction of e. coli promoter sites. volume 20, pages 4331–4338. Oxford Univ Press, 1992.

[10] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. 2015.

[11] David R Kelley, Jasper Snoek, and John L Rinn. Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks. Cold Spring Harbor Lab, 2016.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Jack Lanchantin, Ritambhara Singh, Zeming Lin, and Yanjun Qi. Deep motif: Visualizing genomic sequence classifications. 2016.

[15] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. 2015.

[16] Philip Machanick and Timothy L Bailey. Meme-chip: motif analysis of large dna datasets. volume 27, pages 1696–1697. Oxford Univ Press, 2011.

[17] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. pages 1–23. Springer.

[18] Anthony Mathelier, Oriol Fornes, David J Arenillas, Chih-yu Chen, Grégoire Denay, Jessica Lee, Wenqiang Shi, Casper Shyr, Ge Tan, Rebecca Worsley-Hunt, et al. Jaspar 2016: a major expansion and update of the open-access database of transcription factor binding profiles. page gkv1176. Oxford Univ Press, 2015.

[19] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. page 032821. Cold Spring Harbor Labs Journals, 2015.

[20] Manu Setty and Christina S Leslie. Seqgl identifies context-dependent binding signals in genome-wide regulatory element maps. 2015.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. 2013.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. volume 15, pages 1929–1958, 2014.

[23] Gary D Stormo. Dna binding sites: representation and discovery. volume 16, pages 16–23. Oxford Univ Press, 2000.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[25] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[26] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. volume 12, pages 931–934. Nature Publishing Group, 2015.