Predicting DNA Recombination Attachment Sites

Matt Durrant **Josh Wolff** **Vinay Sriram**

Abstract

Bacterial recombinases mediate site-specific integration events, efficiently adding large segments of new DNA sequences into a target DNA sequence. Understanding how these recombinases work could help us to develop new genome editing tools, making it possible to add novel segments of DNA into the human genome, a task that is currently quite difficult. Our goal was to leverage a recently generated dataset of thousands of bacterial recombinases with predicted binding sites. We first used a convolutional neural network to classify DNA sequences as candidate attachment sites or random DNA sequences. This model performs well, with up to 89.8% test set accuracy. We then built a model to predict which recombinase can bind to each attachment site, and reached 80.3% test accuracy. These promising results demonstrate an exciting and unique application of deep learning.

1 Introduction

Integrative mobile genetic elements are ubiquitous in nature. Often referred to as "jumping genes", these are DNA sequences that carry the molecular machinery necessary to mobilize and insert themselves throughout the genome (1). Some of these elements randomly integrate into the genome, while others integrate only at specific sites. The goal of this project is to use deep learning to better understand the mechanism by which these site-specific recombinases recognize their cognate DNA attachment sites. This can help researchers to develop these recombinases for human genome editing.

In the research lab of Dr. Ami Bhatt, Matt Durrant (an author of this project) has developed a computational comparative genomics pipeline that has generated a database of thousands of putative recombinases, together with their cognate attachment sites (attP and attB). Our goal is to be able classify binding sites as attP, attB, or other (negatives), and to also predict if a given protein can bind to a given site. This should enable us to understand the recombination mechanism in greater detail, taking us one step closer to using such recombinases as general-purpose genome editing tools.

1.1 Problem Formalization

Task I: attX Classification: Task 1 consists of two subtasks. The objective of Subtask 1 is to develop a binary classifier that can determine whether an input attachment site is either attB or attP, while the objective of Subtask 2 is to develop a ternary classifier that can determine whether an input attachment site is attB, attP, or neither (a random input sequence). Note that an attachment site is simply a string of 101-150 nucleotides, each of which is a character in the set $\{A, T, G, C\}$.

Task II: attX Protein Matching: The objective of Task II is to develop a binary classifier that can determine whether or not a given protein binds a given attachment site. The two inputs to the model are (1) the input attachment site (can either be attP or attB) and (2) an input protein sequence. A protein is a sequence of 200-1700 characters, each of which represents one of 20 amino acids.

2 Related Work

Deep Learning in genomics research has become prevalent in recent years (2) (3). The problem that we are tackling is quite unique, but there are similar problems that are being actively researched in the space. One of the most common is to predict where certain DNA-binding proteins (e.g. transcription factors) will bind based on data collected from a controlled experiments such as ChIP-seq (4) (5).

Alipanahi et. al. developed DeepBind, a comprehensive deep learning model based on convolutional neural networks to predict the binding sites of DNA binding proteins (6). The researchers used sequences derived from several different experimental methods that are used to identify binding sites, such as ChIP-seq data. Research performed by Anshul Kundaje's group here at Stanford to focuses on several problems in the field of genomics (7), including a method for decomposing deep learning output predictions called DeepLIFT (8). Quang and Xie built a hybrid convolutional and recurrent deep network to capture both short-term and long-term interactions between DNA sequences (5).

3 Dataset and Features

Matt Durrant has recently developed a computational pipeline to mine public databases containing >100k bacterial genomes to identify recombinases and their predicted attachment sites (9). This pipeline identified 9,000 recombinases, along with predicted attachment sites. This dataset was the starting point of this project. Negative examples were synthesized using various techniques. First, each attachment site DNA sequence was randomly shuffled to produce a new, random sequence. Second, we randomly sampled real sequences from public databases of bacterial genomes. Since negative examples were cheap to synthesize, two-thirds of all examples were negative when training. We trained models on two datasets - the raw/noisy dataset (uncurated), and a cleaned dataset that was less than half the size of the raw dataset. We used a 70-10-20 train-dev-test split. For task 1, subtask 2, the raw dataset contained 177k examples, and the cleaned dataset contained 90k examples. For task 2, the raw dataset contained 231k examples, and the cleaned dataset contained 107k examples.

4 Methods

4.1 Model Ingredients and How They Work

For both tasks described above, combinations of the following ingredients were used in the process of developing the various classifiers. We built and trained all models using the layer, loss, and optimizer abstractions provided by TensorFlow Keras in Python 3.

Densely Connected Layers: A dense layer consumes an input $x \in \mathbb{R}^{n_i}$ and produces an output $y \in \mathbb{R}^{n_j}$ via $y = g(Wx_i + b)$. Note that $W_i \in R^{n_i \times n_j}$ and $b \in \mathbb{R}^{n_j}$ are model parameters and g is some non-linear activation (we use ReLU for this project for all layers except the final activation).

Convolutional Layers: A convolutional layer's parameters include a set of filters that are applied to multidimensional input volumes. Specifically, for a c-channel input volume, f different c-channel filters are convolved to produce an f-channel output volume. The output dimensions are a function of the input dimensions, the filter size, the stride, padding, and max pooling (subsampling the volume). Typically, after some number of convolutional/pooling layers, a dense layer is applied to consolidate the output into vector form. Useful properties of convolutional layers include parameter sharing and translation invariance. For these reasons, convolutional neural networks (CNNs) have historically worked well for genomics research applications; often, the absolute position of a motif (i.e. in a DNA sequence) is less important than its presence and relative position.

Optimizer, Activation and Loss Functions: For binary classifiers, the output of the model is a single probability p generated by a sigmoid activation after the final layer, given by $\sigma(x) = \frac{1}{1+e^{-x}}$. For N-class models, the output is a distribution $p = (p_0, p_1, p_2)$ with the softmax activation. We use the cross entropy loss ($N = 2$ for binary classification and $N = 3$ for ternary classification).

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_i}} \qquad\qquad J = -\frac{1}{N}\sum_{i}^{N} \mathbf{y}_i \log(\hat{y}_i)$$

Finally, we have selected the Adam (adaptive moment estimation) optimizer (10) to use during training as it enjoys the benefits of both the AdaGrad (adaptive gradient) and RMSprop optimizers.

4.2 Architecture and Implementation

4.2.1 Task I: attX Classification

For task I, the baseline model used only dense layers. We created a 4-dimensional one-hot encoding for each of the nucleotides, thereby producing an input matrix $x \in \{0,1\}^{4\times M}$, where M is the maximum attachment site length. We flattened this matrix into a input vector of dimension $4M$, and then passed it through a two dense-layer neural network with ReLu activation functions. We used the cross entropy loss function described in the above section to generate class probabilities.

We achieved improved performance using a CNN. For the CNN, we directly fed the two dimensional input matrix (a single channel volume) into Conv2D layers. Our CNN consisted of a Conv2D layer (64 filters, 3×3 size, same padding, ReLu activation), followed by another Conv2D layer (32 filters, 3×3 size, same padding, ReLu activation), a max pool layer (2×2), two more Conv2D layers (64 filters, 3×3 size, same padding, ReLu activation), and a final max pool layer (2×2). We then passed our flattened CNN output to a 512 unit dense layer (with ReLu activation), followed by a softmax classification layer.

4.2.2 Task II: attX-Protein Matching Task

For task II, we compared two different methods (we refer to them as V1 and V2) for processing the input protein sequence. Note that both models share the convolutional architecture used to encode the attX attachment site.

The V1 model embeds the proteins using an LSTM (11). Recall first that the input protein sequence is simply a string of characters, where each character corresponds to an amino acid. There are 20 unique amino acids. Thus, the V1 model first replaces each character in the protein string with a one-hot vector (20 dimensional, for the 20 possible amino acids). Therefore each protein is represented as a sequence of up to 1700 (max protein length) 20-dimensional vectors. This sequence is fed into a many-to-one LSTM that generates a 100-dimensional encoding of the entire input protein sequence.

By contrast, the V2 model uses the k-mers in order to embed the protein. For a given input protein sequence and selection of k, (for our expleriments, we used $k = 4$), this model first generates a list of all possible k-mers l. Note that a k-mer is just a subsequence of length k. Thus, the protein $avvwlm$ contains the following list of 4-mers: $l = (avvw, vvwl, vwlm)$. We wish to encode which of these k-mers are present in the protein. Because there are a total of 20^k possible such subsequences (160,000 for just $k = 4$), a one-hot representation is computationally intractable. Therefore, we use minhashing as a mechanism for developing a condensed 100-dimensional encoding for each protein. Specifically, each protein is given a minhash signature $s^{[l]} = [s_1, ..., s_{100}]^T$, where $s_i^{[l]} = min_{\pi_i}(l)$. Here, π_i represents a random permutation of the 20^k k-mers, and $min_{\pi_i}(l)$ denotes the lowest index (under the permutation π_i) k-mer present in the list l. Note that minhashing has a nice property. Let $J(l_1, l_2)$ denote the Jaccard similarity of two proteins l_1 and l_2. We have:

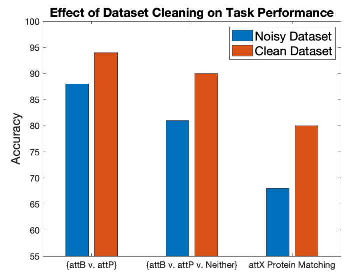
$$J(l_1, l_2) = \frac{l_1 \cap l_2}{l_1 \cup l_2} = Pr\{s_i^{[l_1]} = s_i^{[l_2]}\}$$

The above equation states that the probability that the two minhash signatures agree on any given entry **is the Jaccard similarity of the two proteins**. Therefore, with 100 independently generated permutations, the 100-dimensional minhash embeddings are pairwise similar with respect to Jaccard similarity in the same way that the one-hot encoding vectors of the two proteins are pairwise similar. There is therefore substantial structural information encoded in the minhash signatures.

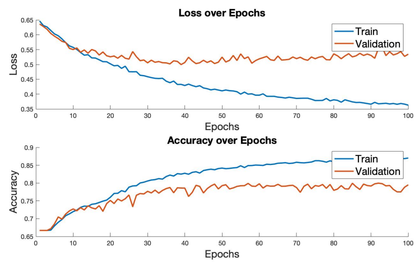
5 Experiments, Results and Discussion

5.1 Hyperparameter Optimization

Between the two tasks tasks, we tuned the following hyperparamters: learning rate, type of input padding, CNN kernel size, layer dimensions, dropout regularization parameter, and L2 regularization parameter. We observed the following. **[1]** Padding the zeros on the right end of the sequence versus using symmetric zero-padding (i.e. splitting the zeros on the left and right sides) resulted in an



(a) Effect of Data Cleaning



(b) Train/Dev Loss and Accuracy (Task II)

Figure 1: Empirical Results

improved development set accuracy between 1% and 4% for our dense neural network. There was no significant improvement for the CNN, likely due to the translation invariance property. **[2]** For task 1, despite our hypothesis that 4×4 kernels might perform better on our 4-d one-hot inputs, we found that 3×3 kernels achieve between 0.5% and 1.0% higher accuracy. **[3]** For task 2, higher dimensional dense layers (i.e. 1024 neurons vs. 512 neurons) required commensurately higher dropout probability (i.e. 0.2 vs 0.1) and L2 reguralization coeffient (i.e. 002 vs. 0.001) to avoid overfitting. Note that for both tasks, early stopping was also used at the peak validation accuracy to regularize.

5.2 Effect of Dataset Cleaning

Plot 1a contains the top test accuracy for each task when trained on different datasets. One key result is that across tasks, despite the fact that the clean dataset contains only roughly half of the data in the original (raw/noisy) dataset, the models trained on the clean dataset outperform those trained on the original. Another key result is that the higher the task complexity, the less robust the models are to noise. Switching from the noisy dataset to the clean dataset produced a 6.8% increase in test accuracy for Task I Subtask 1 (lowest complexity), an 11.1% increase in test accuracy for Task I Subtask 2 (medium complexity), and a 17.6% increase in test accuracy for Task II (highest complexity).

5.3 Task I Quantitative Results

		Predicted		
		attB	attP	None
Truth	attB	0.68	0.01	0.30
	attP	0.01	0.82	0.17
	None	0.02	0.02	0.96

		Predicted		
		attB	attP	None
Truth	attB	0.89	0.01	0.07
	attP	0.01	0.90	0.04
	None	0.11	0.09	0.89

Table 1: This table summarizes the best task I (subtask 2) model performance. The proportions shown are normalized by row for the left table and normalized by column for the right table. For example, 68% of the true attB sequences were correctly identified as attB by the model (left), and in all cases where the model predicted attB, it was correct 89% of the time (right).

Best Model Performance: We achieved **97.7%** training set accuracy and **93.1%** test set accuracy for subtask 1, and we achieved a **94.0%** training set accuracy and **89.2%** test set accuracy for subtask 2 using the CNN model. This represents a marginal improvement over the baseline test set accuracy for subtask 1 (92.5%) and substantial improvement over the baseline for subtask 2 (82.8%). Table 1 above contains results for task I, subtask 2 specifically. We notice that the model is able to robustly discriminate between attP and attB sites. Out of 3,167 examples of attP, the classifier only misclassified 33 of them as attB. Similarly, out of 3,387 examples of attB, the classifier only misclassified 20 of them as attP. Furthermore, when the classifier misclassified the sequences that were neither attB nor attP, it misclassified it as attP and attB with approximately equal frequency, indicating that it is not biased towards either. Thus, we see that the primary source of error occurs in differentiating a valid attachment site from a random sequence. This result is intuitive, as nearly any sequence can potentially exist in the genome. Finally, the class imbalance (3,167 attP and 3,387 attB with 13,108 negatives) does not appear to skew the performance. Table 1 indicates that the model does not blindly predict "neither"; in fact, the probability of a correct classification appears independent to the predicted class.

5.4 Task II Quantitative Results

		Predicted	
		Match	Non-Match
Truth	Match	0.22	0.11
	Non-Match	0.08	0.58

Accuracy	80.37%
Precision	72.82%
Recall	65.62%

Table 2: This table summarizes the best task II model performance. Note that in the confusion matrix, the proportions of the four classes are with respect to the total number of test set examples.

Best Model Performance: We achieved **90.1%** training set accuracy and **80.3%** test set accuracy for the matching task using the minhash-based embedding model (V2). This substantially outperforms the LSTM-based embedding model (V1); the latter only had **71.8%** training set accuracy and **70.1%** test set accuracy. Clearly, the information regarding Jaccard similarity between proteins encoded in the minhash signatures produced a more valuable set of features than the LSTM embedding. We suspect that the cause of this is lack of high data volume. In order to learn character-level LSTM embeddings from scratch, we would likely need on the order of millions of examples, as is common in NLP tasks. Overall, the best V2 model accuracy represents substantial improvement over naive majority class guessing (66.7% accuracy). Finally, Table 2 illustrates that the model's precision is roughly 7% better than its recall (i.e. it is slightly more likely to reject a match rather than accept a non-match). In practice, due to the high verification cost of a potential match, high precision is desirable, as we wish to minimize the number of accepted false candidates.

5.5 Qualitative Results

There is no real sense of "human-level" accuracy in this task, given that discovery of new attachment sites is largely a slow, empirical process with substantial trial and error. We have constructed a class activation map for an example input sequence to task 1. The first layer activation demonstrates that the model learned to minimally activate in regions where there is no substantial data (i.e. padding zones). This suggests that the CNN did in fact robustly learn to ignore certain regions dynamically depending on whether or not they provide relevant information.

6 Conclusion/Future Work

We have shown that deep learning methods can in fact predict recombination attachment sites with minimal preprocessing of the input sequences. For the first task, we reached a test set accuracy that exceeded 89%, with precision exceeding 89% for all three classes. This exceeded our expectations, and the model as it exists currently is quite valuable.

In consulting with an expert in deep learning for genomics, Anshul Kundaje, we were informed that our second task would be considerably more difficult than the first. We found that the second task was indeed more difficult, but we were still observed a significant improvement over random with our best model. Another interesting finding of this study was the importance of cleaning the initial dataset - we substantially increased performance despite cutting the size of the dataset in half.

The model that we have developed for the first classification task will be useful for Matt Durrant's PhD Thesis going forward. It will be used to screen predicted attachment sites, which can then be experimentally validated and developed as novel genome editing tools.

The second task, predicting if protein binds to a given attachment site using primary sequence alone, was more difficult, but it was a good start. Future models could draw upon biophysical homology models of the 3-dimensional structure of the protein in an attempt to improve binding prediction.

7 Contributions

Matt Durrant - Conceptualized the project, generated and curated the data set, established a classification baseline (logistic regression), wrote reports and helped create poster.

Josh Wolff - Trained and refined the various CNN models for the attachment site classification task, wrote reports and helped create poster.

Vinay Sriram - Trained and refined the various CNN models for the protein binding classification task, wrote reports and helped create poster.

References

[1] Smith, M. C. M. Phage-encoded Serine Integrases and Other Large Serine Recombinases. Microbiol Spectr 3, (2015)

[2] Zou, J. et al. A primer on deep learning in genomics. Nat. Genet. 51, 12–18 (2019)

[3] Wainberg, M., Merico, D., Delong, A. Frey, B. J. Deep learning in biomedicine. Nat. Biotechnol. 36, 829–838 (2018)

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

[5] Quang, D. Xie, X. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucleic Acids Res. 44, e107 (2016)

[6] Alipanahi, B., Delong, A., Weirauch, M. T. Frey, B. J. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat. Biotechnol. 33, 831–838 (2015)

[7] Avsec, Ž. et al. Deep learning at base-resolution reveals motif syntax of the cis-regulatory code. bioRxiv 737981 (2019) doi:10.1101/737981

[8] Shrikumar, A., Greenside, P. Kundaje, A. Learning Important Features Through Propagating Activation Differences. in Proceedings of the 34th International Conference on Machine Learning - Volume 70 3145–3153 (JMLR.org, 2017).

[9] Durrant, M. Li, M. Siranosian, B., Bhatt, A. S. (in press), A bioinformatic analysis of integrative mobile genetic elements highlights their role in bacterial adaptation. Cell Host Microbe, preprint on bioRxiv - doi: https://doi.org/10.1101/527788

[10] Ruder, S. (15 Jun 2017). "An overview of gradient descent optimization algorithms." https://arxiv.org/pdf/1609.04747.pdf

[11] Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV. Sequence to sequence learning with neural networks. In NIPS, pp. 3104-3112, 2014.