MotifGAN: Generating Transcription Factor Binding Site Motifs through Generative Adversarial Networks

Anvita Gupta
Department of Computer Science
Stanford University
Stanford, California
`avgupta@cs.stanford.edu`

Abstract

Generative models have been used to great success in computer vision problems to generate life-like images; they have the added capability of being able to learn extremely expressive and concise representations of datasets. However, several more recently developed generative models, such as Generative Adversarial Networks and Variational Autoencoders, have not been applied to problems in genomics. One particular problem in genomics which is of interest is the problem of predicting sequences which can bind to transcription factors, proteins that regulate gene expression. Here we present MotifGAN, a generative adversarial network, and show that MotifGAN makes strides in producing genomic sequences that are significantly enriched for transcription-factor binding site motifs. We deal specifically with the CTCF binding site motif, and additionally present tricks and optimizations for training GANs on discrete genomic data.

1 Introduction

Generative models represent a new class of models which seek to estimate $P(Y, X)$, or the joint probability of the class and the observed data Jordan [5], as opposed to traditional discriminative models, which estimate $P(Y|X)$. By modelling the join distribution, generative models can produce high-probability (X, Y) pairs. Indeed, generative models, like Hidden Markov Models have been used extensively in genomics for applications like generating pairwise alignments, multiple sequence alignments, and gene predictions. However, few of the most recently developed generative models, such as Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs) have been applied to genomics.

Applying GANs to generate sequences containing transcription factor binding site motifs has three main benefits: the first is that since generative models use a low dimension parameter space to generate higher dimensional datapoints, the models are forced to learn a compact and expressive representation of the data. These highly expressive, compact features can then be used to improve the accuracy of discriminative models. Furthermore, genomic data has a great number of unlabeled data points, or datapoints with labels somewhat "artificially" imposed on them (e.g. if 50% of a region overlaps with a TF binding site, that region is classified as "positive" but if 49% overlaps, that region is classified as "negative"). Generative models have been used extensively for unsupervised learning Radford et al. [8] and have recently been modified for semi-supervised learning Salimans et al. [9]. Lastly, novel functional genomic sequences created by generative models have interesting uses in synthetic biology, such as generating sequences that can bind to a specific subset of transcription factors.

In this project, we focus primarily on training GANs to generate sequences with the CTCF binding site motif Consortium et al. [2]. This project represents an exploration of how GANs can be applied to genomics, a novel domain in which they have not yet been investigated.

1.1 Related Work

1.1.1 Generative Adversarial Networks

A GAN consists of two smaller models, a Generator G and a Discriminator D. In the context of computer vision problems, G generates new images from a vector of noise, and D classifies those images as real or fake. The end goal of G is to produce images so realistic that D is unable to classify them as fake. Each pass through the network includes a backpropagation step, where the parameters of G are improved so the generated image looks more realistic. Thus, G and D are playing a minimax game with the following loss function [3]:

$$\min_{G}\max_{D} V(D,G) = \mathbf{E}_{x \in P_{data}(x)}[log(D(x)] + \mathbf{E}_{z \in P(z)}[log(1 - D(G(z))] \tag{1}$$

Concretely, we seek to maximize the probability $D(x)$ that x is real when x comes from a distribution of real data, and minimize the probability that the data point is real, $D(G(z))$, when $G(z)$ is the generated data from some input vector z.

It is worth noting that this loss function does not have a closed form, and consequently a GAN does not necessarily converge– it is generally more unstable than other generative models such as Variational Autoencoders. However, Alex Radford has achieved greater stabilization in DCGAN through Batch Normalization, Leaky ReLUs, and a few other architectural constraints [8]. Radford also presents the idea of generating images based on a "code", or a vector of 100 numbers drawn from the standard normal distribution. This code can be incrementally changed to modify the output of the generator.

1.1.2 Transcription Factor Binding Problem

The problem which we seek to solve here is the problem of generating sequences which bind to transcription factors, proteins which regulate the expression of certain genes, as described in Slattery et al. [10]. Specifically, given a sequence of length n and a particular transcription factor, the discriminator portion of our GAN will predict whether the transcription factor (TF) can bind to the sequence.

Deep convolutional and recurrent neural networks have shown great success in the transcription factor binding site prediction problem. Specifically, convolutional models like DeepBind Alipanahi et al. [1] and DeepSea have been able to classify whether sequences bind to different transcription factors with median auROCs of above 0.95 Zhou and Troyanskaya [12]. DanQ was further able to improve on DeepSea's results by combining DeepSea's convolutional layers with an additional recurrent layer, which allows the network to not only learn specific motifs, but also a "grammar" of how those motifs work together Quang and Xie [7]. Finally, DeepMotif also represents a another successful architecture which has been applied to the TFBS classification problem, as their convolutional neural network achieves a median auROC of 0.890 on the DeepBind dataset used by Alipanahi *et al* [6].

1.2 Dataset and Preprocessing

The main dataset consisted of ten thousand 64bp sequences containing a deterministic, identical motif drawn from the CTCF probability weight matrix. The motif, which was 20bp long, was embedded at random positions throughout the 64bp long sequences.

The simulated sequences were one-hot-encoded for input into the first model architecture. However, the one-hot-encoding requires the Generator to learn complex dependencies in the data, such that if a particular position is predicted to be 'A', that position cannot contain any other nucleotide.

To explore how to solve this problem, a second GAN architecture was trained with a linear encoding of this data. In the linear encoding, the interval from 0-255 was divided into four bins of size 64. Numbers in the first bin were assigned to 'A', numbers in the second bin were assigned to 'C', third to 'G', and last to 'T'. Each base pair from the original sequence was randomly assigned a value from its bin, thus transforming the discrete sequence data to continuous data. This encoding does introduce bias into the model, as, for example, 'A' is placed arbitrarily close to 'C'. The discussion section contains some suggestions for future work as to how this bias may be reduced.

This linear encoding was applied to 10k simulated sequences of 256 bp that contained the CTCF motif. In addition, the encoding was tested on ten thousand 200bp long sequences which experimentally bound to CTCF. This dataset came from the DREAM Challenge [2]. These experimentally verified sequences were flanked with 28bp on each side, bringing the total sequence length to 256bp.

We randomly split the datasets into a training and validation set, reserving ninety percent of the data for the training set and ten percent of the data for the validation set. The use of the validation set for the GAN is a new application; we use the validation set only to estimate the performance of the discriminator after every epoch, not to stop the model's training early. Every GAN is trained for 25 epochs.

2 Methodology

2.1 Architecture 1: Vanilla GAN

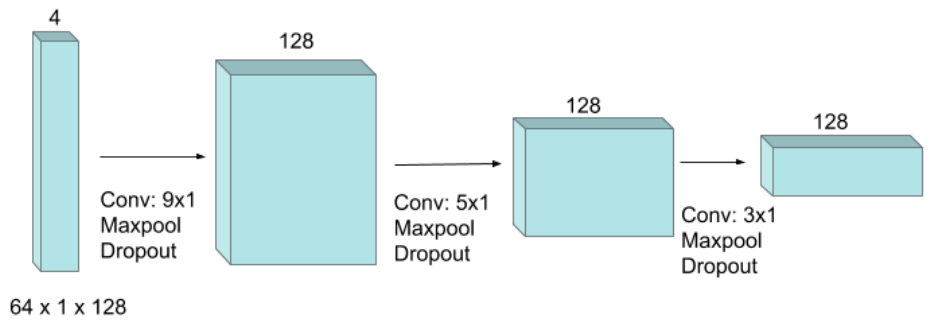

Figure 1: Discriminator Architecture for Vanilla GAN and MotifGAN.

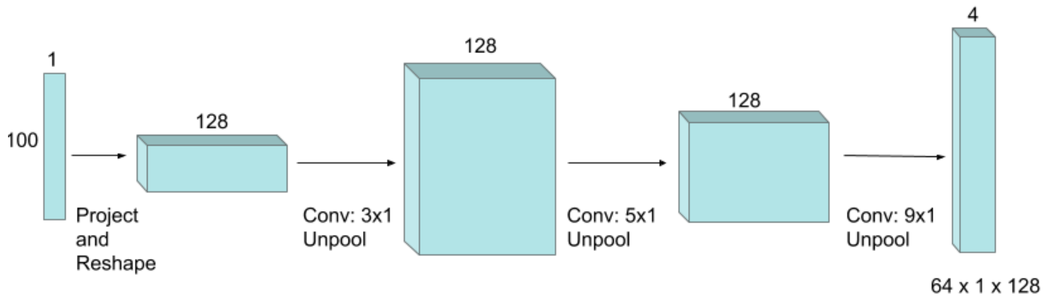

Figure 2: Generator Architecture for Vanilla GAN and MotifGAN.

The architecture of the discriminator is based on DeepMotif [6] and shown in 1. Specifically, the architecture consists of three 1D Convolutional layers, the first with a filter size of [9,1], the second with a filter size of [5,1], and the last with a filter size of [3,1], each with a stride of 1. Each convolutional layer is followed by a leaky ReLU activation layer and a maxpooling layer with filter size [2,1] and stride of 2. The discriminator is also regularized, with dropout after each maxpool layer with $p = 0.3$. The discriminator is trained with minibatch gradient descent, with batch size of 64. The learning rate for the discriminator is 0.02 and the optimizer is the Adam optimizer. The final activation layer of the discriminator was a sigmoidal activation function.

The generator architecture essentially follows from the inverse of the discriminator and shown in 2. The generator takes in a vector $z\ N(0, 1)$ where z has dimensions $100x1$. z is projected and reshaped into a tensor with dimensions $10 \times 1 \times 128$. We then applied a fractionally-strided deconvolution with filter size of [3,1] on this tensor. For more detail on fractionally-strided deconvolutions, we refer the reader to []. Each deconvolution was followed by a ReLU and a 1D unpooling layer, which was implemented according to the strategy in []. The final activation layer of the generator was $tanh$, which normalizes the outputs between [-1, 1]. The output of the generator is a tensor of $64 \times 4 \times 1$, which is intended to represent a one-hot-encoded sequence of length 64. Initially, the output of the generator was not constrained to be discrete, as the goal was to test whether the feedback from the discriminator was enough to push the generator to produce discrete, one-hot encoded output. This architecture did not successfully converge, as seen in the results, as the Discriminator began to predict all examples as negative. Based on the feedback from this architecture, the Discriminator and Generator were modified in two key ways.

2.2 Architecture 2: Ultimate MotifGAN Architecture

The Discriminator's architecture remained the same in this modified architecture, save that the final activation layer was changed from sigmoidal to a softmax activation layer with two outputs. Furthermore, the discriminator was taken out of the GAN and pretrained on our dataset of 10k examples containing the CTCF motif and 10k examples that did not contain the CTCF motif. The weights of the pretrained discriminator were then used to initialize the weights for our discriminator in the GAN. The discriminator was then held constant and the generator was allowed to update.

The Generator's architecture was also modified to ensure that the output from the Generator would be one-hot-encoded. The final activation layer of the Generator was changed to be a softmax layer, followed by a hard-max layer which took the maximum of the four base pairs at every position and set it to 1, and set the other three base pairs to -1. One edge case in this hard-max layer is when the Generator's predictions for all four base pairs were equal (e.g. [0.25, 0.25, 0.25, 0.25]). In this case, the hard-max layer set all four base pairs to 1, which was interpreted as the generator essentially not being sure which base pair was present at that position.

2.3 Architecture 3: Linear Encoding

This architecture was built for sequences that were linearly-encoded according to the strategy described in section 1.2. In addition, this architecture most closely followed the architecture of DCGAN, as described by Radford *et al.* Radford et al. [8]. The Generator's architecture consists of six Deconvolutional layers with filter size of [5,5] and stride of 2. The length and width of the input double after each deconvolutional layer, while the depth is halved. However, instead of expanding z to a size of 64×64 as in DCGAN, in our generator z is expanded to a size of 256×256. The matrix of size 256×256 is then condensed with a single 1d convolution into a matrix of size 256×1.

The discriminator's architecture here is also based off of DCGAN, except that each convolution is a 1d convolution with filter size [5,1] and stride of 2. Thus, the discriminator takes in a sequence of size 256×1 and convolves it to a sequence of size 128×1, 64×1, and so on, ending with a final sigmoidal activation layer.

3 Results

The main metric used to measure the learning of discrminator was its recall:

$$Recall = \frac{TP}{TP + FP} \tag{2}$$

Recall is a more appropriate metric of discriminator performance over time rather than accuracy or ROC AUC, because if the GAN is successful, the accuracy of the discriminator should go down over time. The accuracy goes down mainly because the discriminator begins to classify fake examples incorrectly as real examples. The recall, or proportion of positives (real examples) that the model correctly classifies as posititive, however, should increase over time as the discriminator is trained.

Other metrics of model performance are the loss of the discriminator on real examples, the loss of the discriminator on fake examples, and the loss of the generator. The goal of the generator is simply to minimize the number of fake examples that the discriminator classified correctly, and thus to minimize the loss function in equation 3.

$$Loss_G = \mathbf{E}_{z \in P(z)}[log(1 - D(G(z)))]$$

The final metric of performance for the GAN is whether the sequences sampled from the Generator actually contain the motif of interest, which was contained in every one of the real examples that the discriminator was fed. Since a deterministic motif was embedded, success on this metric was measured by taking a sliding window of 5bp on the motif, and measuring what percentage of the generated sequences contained the base pairs in that window.

3.1 Vanilla GAN

In the Vanilla implementation of the GAN, in which the generator is not constrained to produce discrete output, the discriminator recall on the validation set actually decreases over time, while its loss on real examples increases. However, D's loss goes down on generated examples as shown in Figure 4, meaning that the discriminator correctly classifies more images as fake. Correspondingly, G's loss increases. These results indicate that D here slowly goes from predicting nearly all examples as positive (95% recall) to predicting that all the data points are fake. This is an unusual failure mode, and, at the very least, indicates that discriminator is not performing its job of driving the generator to improve its examples to correspond with real ones.

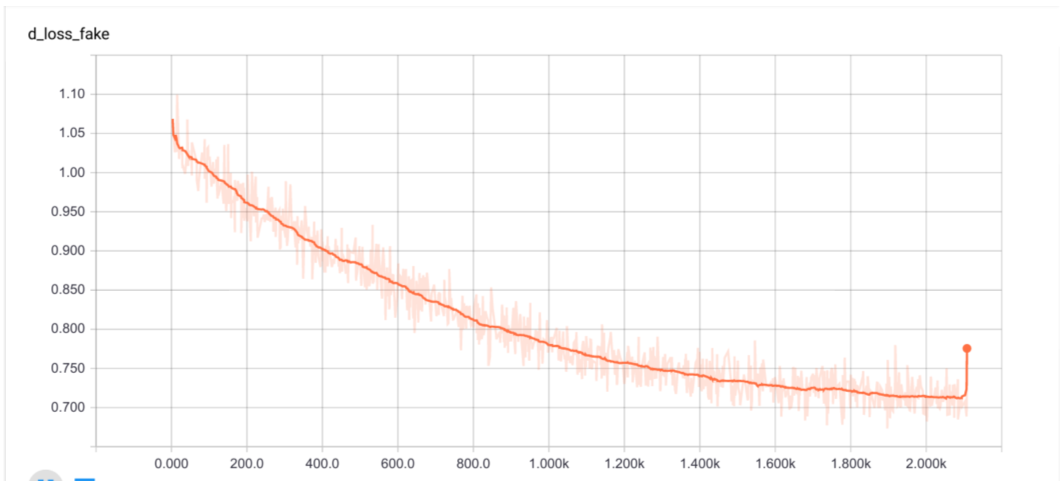

Figure 3: Discriminator loss on fake examples decreases over time for the Vanilla GAN, meaning that the discriminator is becoming more successful at classifying fake examples as fake.

3.2 Pretrained Discriminator

To rectify this problem, we pretrained the discriminator separately on ten thousand simulated sequences containing the deterministic CTCF motif, as well as ten thousand simulated sequences without the CTCF motif. The discriminator reached a validation AUC ROC of 0.95 for this problem, as well as a validation recall of 0.98 after forty epochs. The change in validation recall as training ocfurs is shown in Figure 5. Training was stopped after this point, and these weights were used to initialize the discriminator in our ultimate MotifGAN architecture.

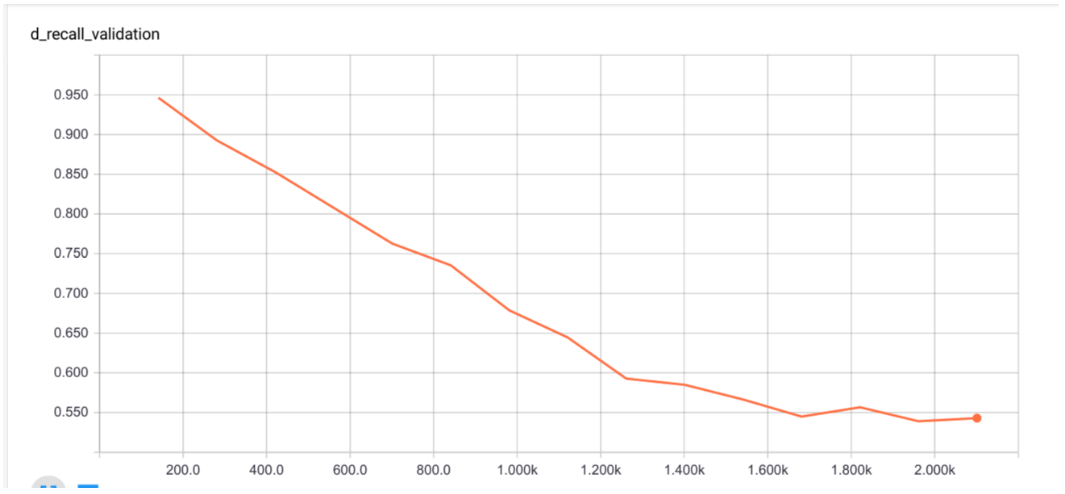

Figure 4: Discriminator recall decreases steadily over time. Implies that Discriminator starts classifying all examples as negative.

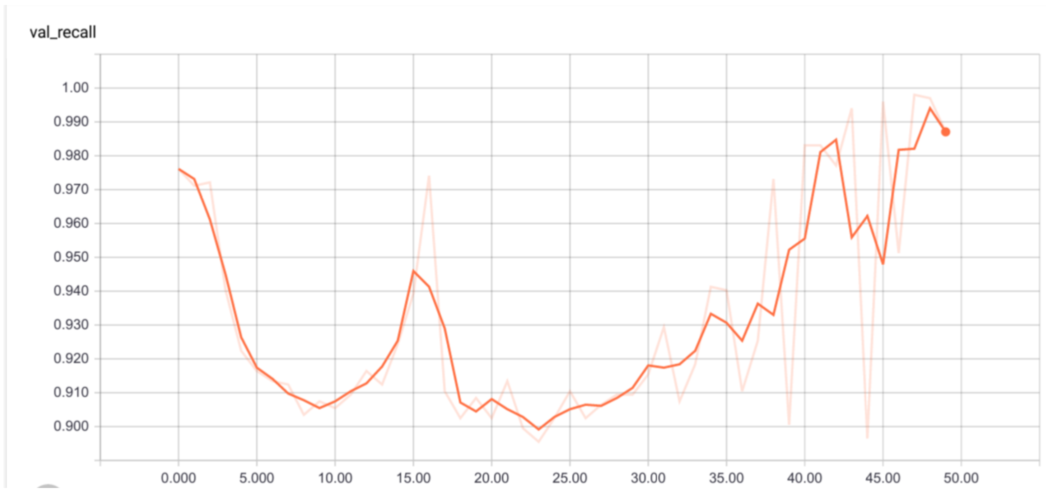

Figure 5: Recall on validation set when discriminator trained alone.

3.3 MotifGAN

As expected and as shown in Figure 6, the loss of the pretrained discriminator on real examples stayed relatively constant over time, as the discriminator's parameters were held constant. Correspondingly, the loss of the generator decreased over time, as shown in 7, beginning at 13.0 and falling by about 50% to 5.74.

We sampled 512 *de novo* generated sequences from batches from the generator that each had a loss of less than 6.5, which was a cutoff that was arbitrarily chosen since it was near the minimum loss that the generator was able to reach. As mentioned earlier, a sliding window of 5bp was created for the CTCF motif, and the percentages of sequences which contained each sliding window is shown in figure 8. The percentages of generated sequences from the initial epoch, when the generator loss was much higher, are also plotted as a comparison. Figure 8 show that the generated sequences for the final epoch are significantly enriched for sections from the original motif, as compared to the generated sequences from the initial epoch. In particular, the generated sequences are enriched with the latter part of the initial motif. Indeed, one hundred percent of the generated sequences contain the quintuple C ('CCCCC') section of the original embedded motif, and eighty-eight percent of the generated sequences contain the motif immediately after the quintuple C, 'CCCCT'.

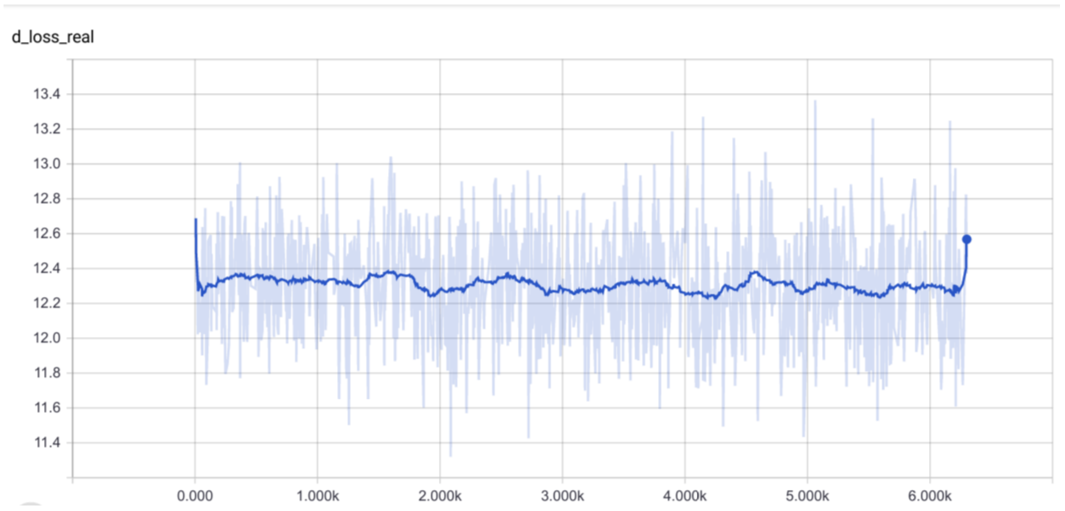

Figure 6: Loss of the pretrained discriminator in MotifGAN on the real examples. As expected, the loss stays relatively constant.

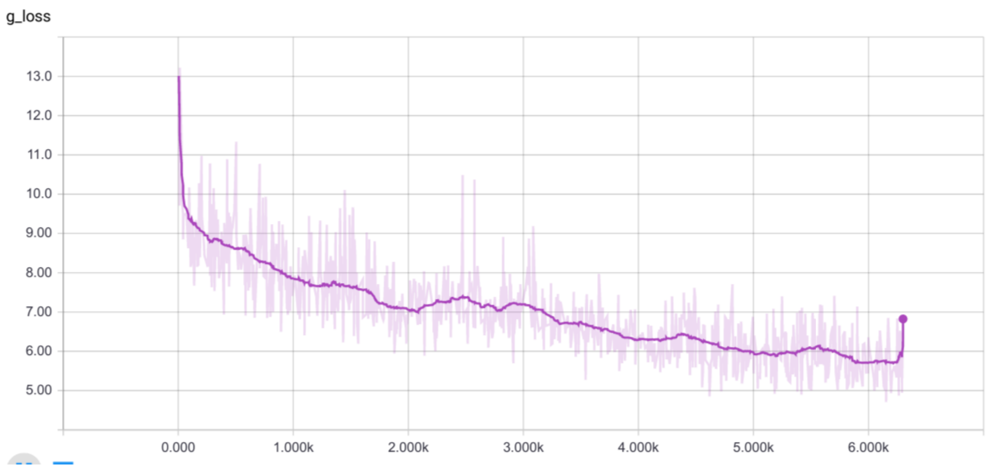

Figure 7: Loss of the generator in MotifGAN, in which the discriminator is already pretrained.

3.4 Feature Importance Scoring in Discriminator

We took the gradient of the output softmax layer of the pretrained discriminator with respect to the input layer to derive which sections of the input sequences were most predictive of a positive classification from the discriminator. Specifically, we sought to determine whether the discriminator (i) picked up the embedded CTCF motif and (ii) if so, whether some section of the 20bp long motif was more predictive of a positive score than the other sections. The gradient*input scores were plotted for the positive sequences, as shown in Figure 9. Plots of the scores show that the discriminator did classify on the basis of the CTCF motif. Furthermore, in most examples the quintuple 'C' in the center of the CTCF motif, and the base pairs that followed it, had greater gradient*input scores than the initial sections of the sequence.

In addition to these qualitative metrics, we also examined the sequences that would be generated if the base pair with the maximal gradient*input score were assigned to every position. We call these the "maximally activated" sequences. Since the discriminator should be classifying sequences based on the presence or absence of the CTCF motif, we would expect the gradient*input scores to be higher for the CTCF motif and thus for the CTCF motif to appear in the maximally activated sequences. Thus, we generated a sliding window of 5bp for the CTCF motif and examined the percentages of maximally activated sequences that contained those motifs.

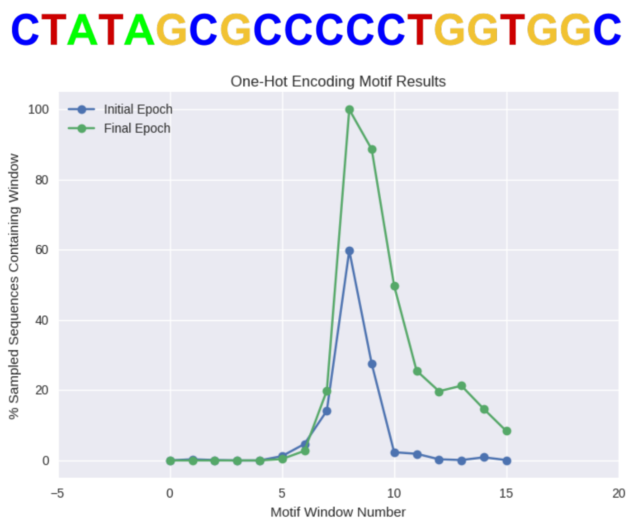

Figure 8: Percentage of generated sequences that contained each window of the motif. The X axis is the window number, which roughly corresponds to the section of the motif displayed above the graph at that location. For example, window 7 on the x-axis corresponds to the 'CCCCC' in the original motif.

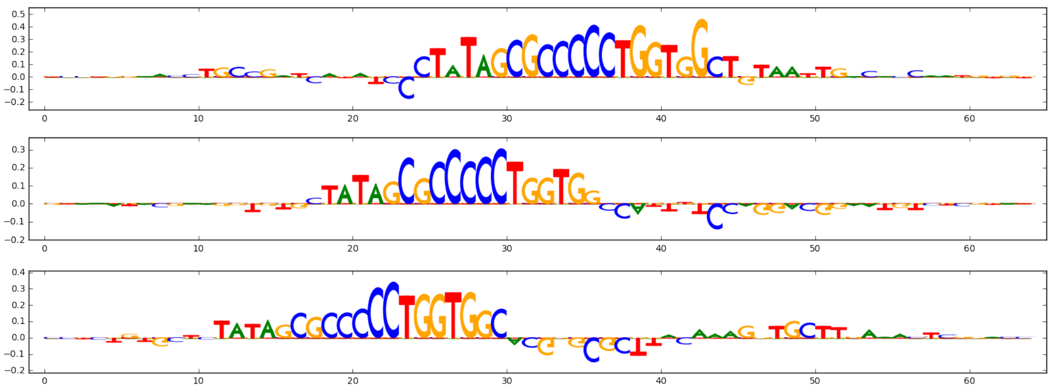

Figure 9: Gradient*input scores for positively-classified sequences.

As shown in Table 1, the quintuple Cs in the center of the motif and the base pairs following it were both the most predictive parts of the motif.

3.5 Linear Encoding Architecture Results

The architecture based off of DCGAN converged extremely quickly for the experimental CTCF-binding dataset, with both the generator and discriminator's loss converging to 0.7 and 1.4 extremely quickly. However, the recall on the validation set for the discriminator fluctuated wildly and did not ever increase beyond 0.5. This was an indication that the discriminator was not accurately able to classify positive sequences, and thus unable to push the generator to create sequences which contained the CTCF motif drawn from the CTCF probability weight matrix.

For the simulated dataset with the deterministic CTCF motif embedded, neither the discriminator nor the generator converged cleanly, as can be seen in Figure 10. The discriminator's loss on the fake datapoints stayed low and relatively constant throughout the training procedure, save for a sudden spike in loss in the twenty fifth epoch, which, as expected, corresponded with a sudden drop in the generator's loss. When examining the generated sequences from the final epoch, as shown in table 3.5, it is evident that parts of the CTCF motif were enriched in the final generated sequences,

Table 1: Percentage of maximally activated sequences that contained 5bp sliding windows of the embedded deterministic CTCF motif.

Motif Window	Percentage
CTATA	11.7647058824
TATAG	20.5882352941
ATAGC	17.6470588235
TAGCG	20.5882352941
AGCGC	20.5882352941
GCGCC	38.2352941176
CGCCC	38.2352941176
GCCCC	41.1764705882
CCCCC	44.1176470588
CCCCT	44.1176470588
CCCTG	38.2352941176
CCTGG	26.4705882353
CTGGT	29.4117647059
TGGTG	26.4705882353
GGTGG	17.6470588235
GTGGC	14.7058823529

although much less so than with the one-hot encoded data. This could be because the linear encoding introduces bias into the results of the model, since ’A’ becomes artificially closer to ’C’, for example. Although this encoding introduces bias, it has been shown that continous encodings are generally more optimal for having GANs converge Yu et al. [11]. Furthermore, we may be able to modify this encoding so that it is more biologically motivated– for example, we may place the purines and pyrimidines closer to eachother in the encoding since purines are likely to mutate to purines and vice versa.

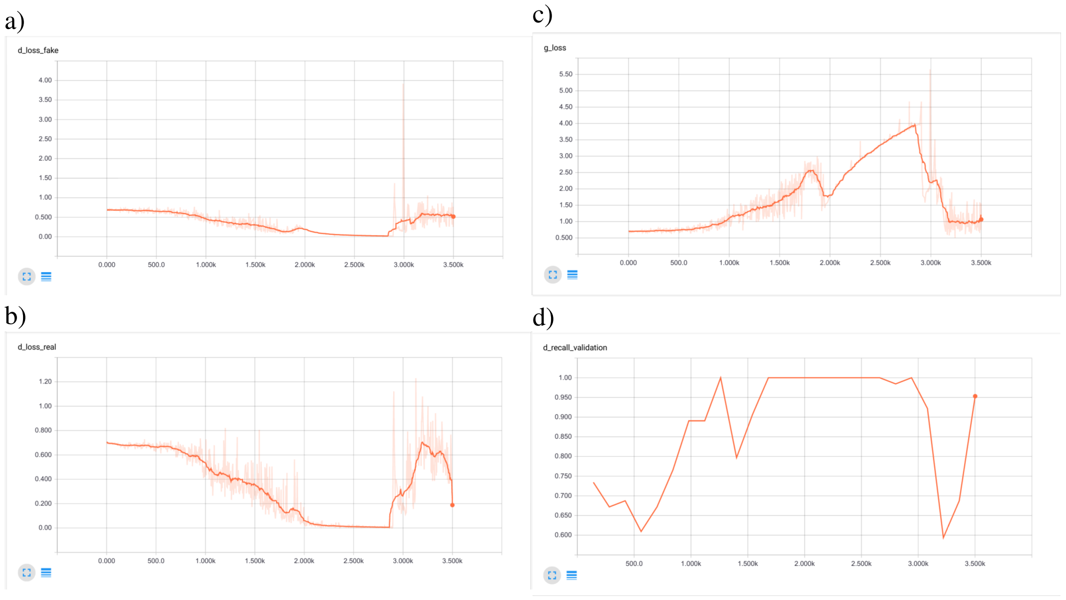

Figure 10: Results for Architecture 3 on linearly-encoded sequences. a) The discriminator loss on the fake examples. b) The discriminator loss on the real examples. c) The generator loss d) The discriminator recall on validation set.

4 Conclusion and Future Work

The sections of the motif that were significantly enriched in the sequences generated from MotifGAN correspond to the sections of the motif that were highly important for the discriminator to classify

Table 2: Percentage of generated sequences from linear-encoding architecture that contained 5bp sliding windows of the embedded deterministic CTCF motif.

Motif	Percentage
CTATA	17.3267326733
TATAG	17.0792079208
ATAGC	11.1386138614
TAGCG	10.1485148515
AGCGC	6.1881188119
GCGCC	8.9108910891
CGCCC	6.6831683168
GCCCC	3.7128712871
CCCCC	2.9702970297
CCCCT	4.4554455446
CCCTG	4.2079207921
CCTGG	4.2079207921
CTGGT	4.2079207921
TGGTG	3.4653465347
GGTGG	6.1881188119
GTGGC	8.1683168317

sequences as positive. This correspondence indicates that MotifGAN is successfully able to detect and recreate sections of the motif that lead the GAN to classify the sequence as positive. In addition, we illustrate the usefulness of particular techniques, such as pretraining the discriminator and forcing the output of the generator to be discrete, in problems in genomics. Without these techniques, we demonstrate that the GAN collapses, consequently producing garbage.

However, this work represents only an initial exploration into the application of Generative Adversarial Networks for genomics problems. There are several possible areas for improvement– most notably, one might investigate how to coax the generator into recreating the entirety of the CTCF motif, which is 20bp long. In order to recreate such a long motif, we might experiment with increasing the filter size for the discriminator. Ensuring that the discriminator is able to pick up the entire motif through the gradient*input scores (or Deeplift scores) is an important first step in ensuring that the generator can create sequences that contain the entire motif.

Another milestone in the application of GANs to this problem is developing GANs that can recreate motifs drawn from a probability weight matrix, rather than deterministically embedded motifs. Indeed, we initially trained the GAN on linearly-encoded motifs that were experimentally determined to bind to CTCF. However, the discriminator itself was not able to achieve a recall of more than 0.5 on this dataset, and consequently the generator was not able to produce sequences enriched with motifs drawn from a PWM.

Finally, we might experiment with applying RNNs to this problem, as RNNs are built to deal with sequential data and have been applied with great success in chromatin accessiblity prediction networks such as DanQ Quang and Xie [7]. Adding a RNN in the discriminator may improve the ability of the discriminator to pickup large motifs, such as the 20bp CTCF motif. Beyond that, the entire GAN could be built with a recurrent architecture, as laid out in [4].

Developing generative models to create DNA sequences that accurately bind to transcription factors could have many interesting applications in synthetic biology, such as creating sequences that bind to specified subsets of transcription factors. Ultimately, this work represents an exploration into a possible application of Generative Adversarial Networks to this problem and related problems in genomics.

References

[1] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat Biotechnol*, 33 (8):831–838, 2015. ISSN 1087-0156. doi: 10.1038/nbt.3300. URL `http://dx.doi.org/10.1038/nbt.3300`.

[2] Roadmap Epigenomics Consortium, Anshul Kundaje, Meuleman, et al. Integrative analysis of 111 reference human epigenomes. *Nature*, 518(7539):317–330, 2015. ISSN 0028-0836. doi: 10.1038/nature14248. URL http://www.ncbi.nlm.nih.gov/pubmed/25693563.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[4] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *CoRR*, abs/1602.05110, 2016. URL http://arxiv.org/abs/1602.05110.

[5] A. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. 2002. URL https://books.google.com/books?hl=en&lr=&id=GbC8cqxGR7YC&oi=fnd&pg=PA841&ots=ZwP6I7-vB3&sig=B22eJKtzXI_CPCpr-nxdQamIEHs.

[6] Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep gdashboard: Visualizing and understanding genomic sequences using deep neural networks. *CoRR*, abs/1608.03644, 2016. URL http://arxiv.org/abs/1608.03644.

[7] Daniel Quang and Xiaohui Xie. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *bioRxiv*, page 032821, 2015. ISSN 0305-1048. doi: 10.1101/032821. URL http://biorxiv.org/content/early/2015/12/20/032821.abstract.

[8] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL http://arxiv.org/abs/1511.06434.

[9] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

[10] Matthew Slattery, Tianyin Zhou, Lin Yang, Ana Carolina Dantas Machado, Raluca Gordan, and Remo Rohs. Absence of a simple code: How transcription factors read the genome. *Trends in Biochemical Sciences*, 39(9):381–399, 2014. ISSN 13624326. doi: 10.1016/j.tibs.2014.07.002.

[11] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL http://arxiv.org/abs/1609.05473.

[12] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–4, 2015. ISSN 1548-7105. doi: 10.1038/nmeth.3547. URL http://dx.doi.org/10.1038/nmeth.3547{\T1\textbackslash}nhttp://www.ncbi.nlm.nih.gov/pubmed/26301843.