A MECHANISTICALLY INTERPRETABLE NEURAL-NETWORK ARCHITECTURE FOR DISCOVERY OF REGULATORY GENOMICS

Alex M. Tseng
tseng.alex@gene.com

Gökcen Eraslan
eraslan.gokcen@gene.com

Nathaniel L. Diamant
diamant.nathaniel@gene.com

Tommaso Biancalani
biancalani.tommaso@gene.com

Gabriele Scalia
scalia.gabriele@gene.com

Biology Research | AI Development
Genentech

ABSTRACT

Deep neural networks have shown unparalleled success in mapping genomic DNA sequences to associated readouts such as protein–DNA binding. Beyond prediction, the goal of these networks is to then learn the underlying motifs (and their syntax) which drive genome regulation. Traditionally, this has been done by applying fragile and computationally expensive post-hoc analysis pipelines to trained models. Instead, we propose an entirely alternative method for learning motif biology from neural networks. We designed a *mechanistically interpretable* neural-network architecture for regulatory genomics, where motifs and their syntax are directly encoded and readable from the learned weights and activations, thus eliminating the need for post-hoc pipelines. Our model is also more robust to variable sequence contexts and against adversarial attacks, while attaining predictive performance comparable to its traditional counterparts.

1 INTRODUCTION

Transcription factors (TFs) are proteins that regulate gene activity by recognizing and binding to specific short DNA sequence patterns—or "motifs"—in regulatory elements of the genome (Lambert et al., 2018; Siggers & Gordân, 2014). High-throughput experiments profile genome-wide regulatory activity in diverse cell types, measuring protein–DNA binding (and associated events) across the genome (Consortium, 2012). In general, the regulatory function of a DNA sequence is defined by the combination of motifs in that sequence. These motif configurations have a soft syntax (i.e. density, spacing, orientation, and co-binders), which induces a particular function such as TF binding.

As such, accurately predicting genome regulation from DNA sequence requires expressive models to capture the complexities of the motifs and their syntax. In recent years, deep neural networks (DNNs) have achieved state-of-the-art performance in mapping DNA sequence to TF binding and associated readouts (Alipanahi et al., 2015; Zhou & Troyanskaya, 2015; Kelley et al., 2016; Žiga Avsec et al., 2021). These models accept DNA sequences as an input, and predict labels measured by a regulatory profiling experiment (e.g. a binary label denoting whether or not a TF bound to that sequence). The common goal of these genomic DNNs is to ultimately identify the fundamental code (i.e. motifs and their syntax) underpinning genomic regulation (Novakovsky et al., 2022).

This goal has inspired several methods which perform post-hoc interpretation and analysis of genomic DNNs. Through extensive computational pipelines, these tools attempt to extract the learned motifs and their syntax from trained DNNs. Although these pipelines have an impressive ability to recover

genome biology, they are also composed of many computationally expensive steps which tend to be unstable and require constant human intervention (Novakovsky et al., 2022; Shrikumar et al., 2018).

In this work, we propose an entirely different method of recovering genome biology from a DNN, based on the relatively new concept of mechanistic interpretability. Our method, Analysis of Regulatory Genomics via Mechanistically Interpretable Neural Networks (ARGMINN), enables motifs and their syntax to be directly readable from the network architecture, without any post-hoc pipelines. Our novel architectural contributions include: 1) a novel regularizer to ensure that the first layer's weights directly encode a non-redundant set of relevant motifs; and 2) a modified attention mechanism which reveals motif instances and their syntax in any sequence with a single forward pass.

2 Related Work

Nearly all DNNs for genome regulation have convolutional filters as the first layer (Alipanahi et al., 2015; Zhou & Troyanskaya, 2015; Kelley et al., 2016; Žiga Avsec et al., 2021). To recover motifs, most works have visualized the filter weights or identified subsequences which maximally activate each filter (Alipanahi et al., 2015; Kelley et al., 2016). Although this has shown some limited success, these methods assume that each filter learns one motif, and each motif is learned by one filter. As we will show, this is generally not true because—without special constraints—motifs are typically learned in a *distributed* fashion, where each motif is learned across many filters and layers.

As a result, more sophisticated post-hoc pipelines were developed to extract motifs from trained DNNs. These pipelines first integrate over the entire DNN to compute importances across the dataset (e.g. via integrated gradients or DeepLIFT), resulting in an importance score at each position for each sequence (Sundararajan et al., 2017; Shrikumar et al., 2017). These importances are then segmented into high-importance regions as putative motif instances. Owing to the extreme noisiness of importance scores, however, these putative instances must be clustered into clean motifs by specific tools, such as MoDISco (Shrikumar et al., 2018). Next, to discover motif syntax, motifs must be scanned across the dataset to call individual motif *instances*. Each step of this pipeline is computationally expensive, and for most datasets, the time taken to recover motifs and syntax is over an order of magnitude more than the time taken to train the model. Furthermore, in order to discover motifs and their syntax, these pipelines heavily rely on importance scores from a black-box model, which can be extremely fragile, as importance scores frequently fail to reveal a model's true decision-making process (Ghorbani et al., 2017; Tseng et al., 2020; Alvarez-Melis & Jaakkola, 2018).

Within the field of explainable AI, much effort has focused on post-hoc explainability, where feature-importance scores or decision rules are extracted from a pre-trained black-box model. More recently, however, there has been some burgeoning work beginning to explore the realm of *mechanistic interpretability*, where the patterns and rules learned by a DNN are retrievable directly from the weights and operations that the model learns (Ghorbani et al., 2019; Conmy et al., 2023; Cammarata et al., 2020; Liu et al., 2023; Barbiero et al., 2023).

Our work is also related to concept bottleneck models (CBMs) (Koh et al., 2020; Ghorbani et al., 2019; Kim et al., 2017). CBMs provide meaningful concept-based explanations, but are very limited by the requirement for pre-defined concepts and concept-labeled inputs (which are labor-intensive to obtain). In our work, ARGMINN can be viewed as a type of CBM, but the concepts are motifs, and *the concepts are learned entirely from the data*, thus alleviating the typical limitations of CBMs.

3 A mechanistically interpretable neural network for genomics

We propose a mechanistically interpretable DNN architecture which is designed to 1) accurately predict regulatory function (e.g. protein binding) from DNA sequence; 2) reveal crucial motifs responsible for function across the dataset; and 3) reveal motif instances and their syntax in any given query sequence. Importantly, in a mechanistically interpretable architecture, 2) and 3) are directly encoded in the model weights or activations.

Our architecture consists of two modules (Figure 1a). Firstly, the *motif scanner* module consists of a single convolutional layer which identifies motifs from the input sequence. We developed a unique regularizer for the filters such that the filter weights directly encode non-redundant motifs, thereby accomplishing goal 2) above. The convolutional activations are passed to the second module—the

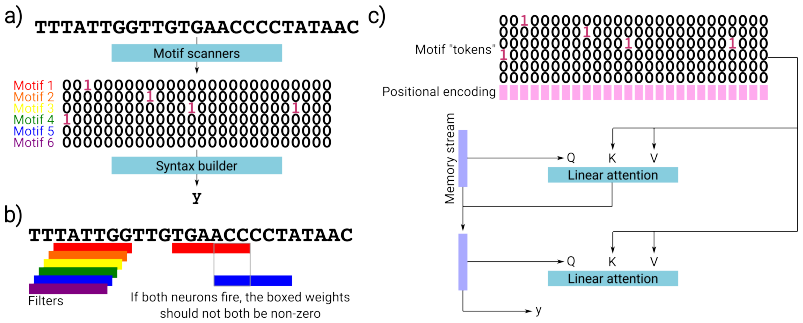

Figure 1: Schematic of the ARGMINN architecture. **a)** The motif-scanner module produces a set of activations denoting which positions match which motifs (the activation magnitude reflects match strength). The activations are passed to the syntax builder, which combines motif instances into a final prediction. **b)** The motif-scanner module is a single convolutional layer. Regularization ensures that each filter learns one motif, and *vice versa*. **c)** The syntax builder is a series of modified attention layers. Each layer derives a single query from the memory stream, and key/value vectors from the original activations, which are used to update the memory stream.

syntax builder—consisting of several layers of modified attention that assemble the syntax and logic between motifs to produce a final prediction. The syntax builder is designed such that the activations and attention scores immediately reveal motif instances and their syntax, thus achieving goal 3).

3.1 Module 1: Motif Scanners

The "motif scanners" are a set of n_f convolutional filters of width w, which are scanned across the input sequence to produce a set of activations (Figure 1b). The network is trained such that the multiplicative weights of each filter directly encode a motif. Thus, a filter's activation is maximized when it scans over a one-hot-encoded sequence that exactly matches the encoded motif. These activations are then thresholded by a ReLU (the additive bias parameter of the filters and the ReLU allow the network to selectively cut off weak matches, thereby producing more sparse activations). For a 1-hot encoded DNA sequence $S \in \{0,1\}^{l\times 4}$, convolutional weights $W \in \mathbb{R}^{n_f \times w \times 4}$ (and bias $b \in \mathbb{R}^{n_f}$) yield activations $A \in \mathbb{R}^{(l-w+1)\times n_f}$:

$$A = \text{ReLU}(\text{Conv}(S, W, b)). \tag{1}$$

Importantly, the filters are regularized so that each filter learns one distinct motif (and each motif is learned by one filter). This allows motifs to be directly read from the filter weights after training. We propose a novel secondary objective which penalizes different filters from activating on overlapping sequences. This is combined with a simple L1-penalty on the filter weights to induce sparse filters that directly reveal *distinct, non-redundant* motifs.

At each position i in a sequence S, each filter aggregates values $S[i, i+w]$. Let $a = \text{argmax}\{A_i\} \in \{0, ..., n_f - 1\}$ be the index of the maximally activated filter at i. For all other filters W_b ($b \neq a$), if W_a and W_b both achieve non-zero activation in the same neighborhood of S, then they should not both have non-zero weights in the overlapping region (i.e. they should not be attending to the same part of the sequence). In other words, every position of S is "protected": at most one filter can activate while attending to any position. If another filter is activated nearby, then its weights should not be attending to the protected part of the sequence. Our *filter-overlap* regularization then can be defined as the following loss function:

$$\mathcal{L}_o(A, W) = \sum_{i=0}^{l-w} \sum_{b \neq \text{argmax}\{A_i\}} \sum_{j=i-(w-1)}^{i+(w-1)} \Big[A_{j,b} \\ \|W_{\text{argmax}\{A_i\}}[\max\{0, j-i\}, w-1-\max\{0, i-j\}]\|_1 \\ \|W_b[\max\{0, i-j\}, w-1-\max\{0, j-i\}]\|_1 \Big]. \tag{2}$$

At each position i, we compare the filter weights of the maximally activated filter W_a with the weights of all other filters (W_b), at every possible overlapping window j. We penalize the L1 norm of the overlapping weights, multiplied by the activation of filter b (i.e. if W_b is not activated, there is no penalty). Importantly, this is a *soft* regularization which the model can choose to ignore if needed for performance. This regularization helps prevent: 1) two filters learning the same motif (or the same part of the same motif); and 2) two filters learning a motif in an interleaved fashion. However, our regularization still allows for a long motif to be learned by two filters, split somewhere down the middle (or similarly, two half-sites directly next to each other, each learned by one filter).

In practice, this loss is computed efficiently by caching all possible windows of weight sums (for each value of j) once, and at each i scaling the window products with the activation of W_b.

3.2 Module 2: Syntax Builder

After the motif-scanner module, positional encodings P are concatenated with the activations A. The second stage of the architecture is the syntax builder, which consists of n_L layers of a modified attention mechanism (Figure 1c). As opposed to a typical attention layer, our modified attention layer has an explicit "memory stream" m_l which is updated after each layer. Each layer derives a *single* query from m_l, resulting in a linear vector of attention scores rather than a quadratic matrix. Additionally, each layer always derives key and value vectors directly from the original "tokens" (i.e. $A \| P$). Each attention layer can be described as follows:

$$\begin{aligned} q_l := W_{q,l} m_{l-1}, K_l := W_{K,l}[A\|P], V_l := W_{V,l}[A\|P] \\ a_l := \frac{K_l q_l}{\sqrt{d_{q_l}}}, m_l := m_{l-1} + \text{MLP}(a_l V_l), \end{aligned} \tag{3}$$

where m_0 is a vector of all 1s, and d_{q_l} is the dimension of the query vector. We also include n_h attention heads, but do not show the reshaping operations above for clarity.

Note that each layer derives a query from the memory stream, and derives key and value vectors from the original activations. Thus, by tracing through the attention scores for any input sequence, we can easily identify which filters/motifs (with their positions) are responsible for a prediction. Each layer is capable of attending to multiple motifs (due to the multiple attention heads), and successive layers allow the model to capture *interactions* between motifs (e.g. with two layers, the model can reason about pairs of motifs, etc.). All together, our final loss function becomes:

$$\mathcal{L}(S, A, W) = \mathcal{L}_{pred}(f(S), y) + \lambda_o \mathcal{L}_o(A, W) + \lambda_l \|W\|_1, \tag{4}$$

4 ARGMINN directly encodes motifs in its weights

The first main advantage of ARGMINN is that—due to the filter-overlap regularization—motifs are directly encoded as filter weights. Over several simulated and real-world experimental datasets, we trained ARGMINN and a standard CNN architecture. We examined the first-layer convolutional filter weights, and systematically matched each filter to the closest known motif (Figure 2a). For simulated datasets, we matched each filter to the closest motif in the simulation; for experimental datasets, we matched each filter to the closest known human motif. In general, ARGMINN was able to encode *known, relevant motifs* directly in the filter weights. Particularly with the experimental datasets, the singular most similar motif (an extremely strict requirement) to an ARGMINN filter is—impressively—typically a known relevant motif. For example, ARGMINN trained on FOXA2 in HepG2 (a pioneer factor) revealed several FOX-family factors, the HNF4 family, and the CEBP family: all have been known to co-localize or co-bind with FOXA (Seachrist et al., 2021; Geusz et al., 2021; Liu et al., 2020). In contrast, traditional DNNs often *distribute* motifs across filters or layers, so methods that try to discover motifs from these architectures by directly analyzing filters can be foiled by the fact that the filters typically do not encode distinct motifs. Furthermore, even when the traditional CNN also encoded motifs in its filters, ARGMINN's filters had significantly higher similarity to the true motifs (Figure 2b).

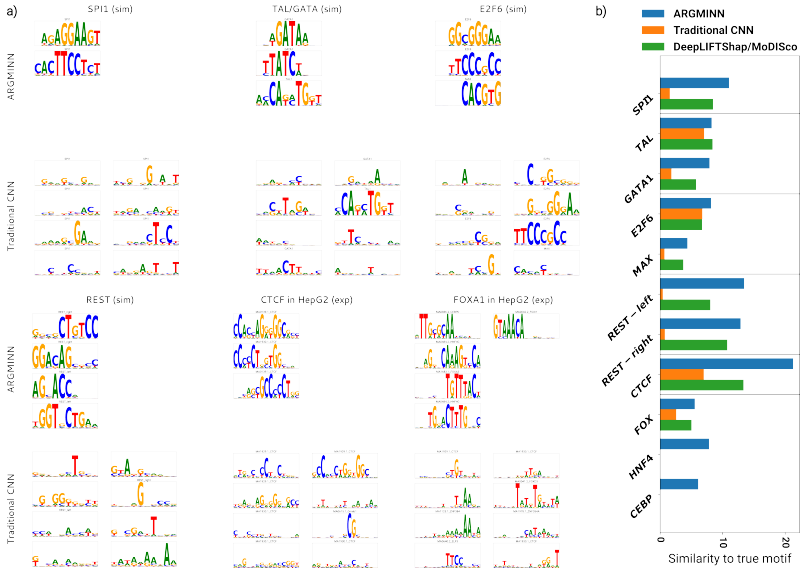

Figure 2: Learned motifs. **a)** We show the first-layer filter weights learned by each architecture. For each filter, we labeled the closest match to a real motif. Filters which never achieve non-zero activations are excluded. **b)** For each relevant motif in each dataset, we show the maximal similarity to any discovered motif by each method. A similarity of 0 means that the motif was not found by that method.

We then compared ARGMINN's motifs to those discovered by an extensive post-hoc pipeline. We computed importance scores from the traditional CNN and used MoDISco Shrikumar et al. (2018) to segment and cluster them into motifs. Whereas ARGMINN clearly revealed motifs as weights, importance scores are generally very noisy and inconsistent approximations of a model's decision making Ghorbani et al. (2017); Hajiramezanali et al. (2023) and they showed only noisy and fragile motif instances (Supplementary Figure S1). As such, MoDISco-identified motifs were almost universally not as close to relevant motifs as those discovered by ARGMINN (Figure 2b). Importantly, they also included many redundancies and extraneous motifs (Supplementary Figure S2).

Finally, to further show the benefit of our filter regularizer, we applied our regularization to a traditional CNN. As a result, the CNN's filters also encoded relevant and non-redundant motifs (Supplementary Figure S3). Importantly, however, without the other architectural novelties of ARGMINN, such a CNN would still not reap benefits such as motif-instance and syntax identification.

5 ARGMINN REVEALS MOTIF INSTANCES/SYNTAX IN ONE FORWARD PASS

ARGMINN reveals motif instances and syntax for any sequence in one forward pass. With previous DNN-based methods, identifying syntax required first learning motifs and then scanning sequences to "call" motif instances. Not only is this computationally expensive, but instance calling by sequence scanning tends to be highly inaccurate (e.g. due to partial hits or missing context which the DNN would have considered).

Instead, ARGMINN reveals motif instances by tracing through the attention scores from a forward pass on any sequence. Because each attention layer derives keys/values directly from the motif-scanner activations, high attention scores directly point to the precise motif instances which the network deems important for its prediction. For example, on our REST dataset, we directly recovered the binding syntax—including spacing preferences—of the two halves of the REST motif (Figure 3a).

We then evaluated the motif instances identified by ARGMINN versus a traditional pipeline. We used FIMO (Bailey et al., 2015) to call instances of the MoDISco-discovered motifs in a set of test sequences. In each case, we ranked the instances by confidence and computed the number of hits in the top k which overlapped the ground-truth instance; for our experimental datasets, we used independently derived binding footprints as ground truth (Vierstra et al., 2020). In general, ARGMINN's motif instances were far more accurate than those found by the baseline (Figure 3b).

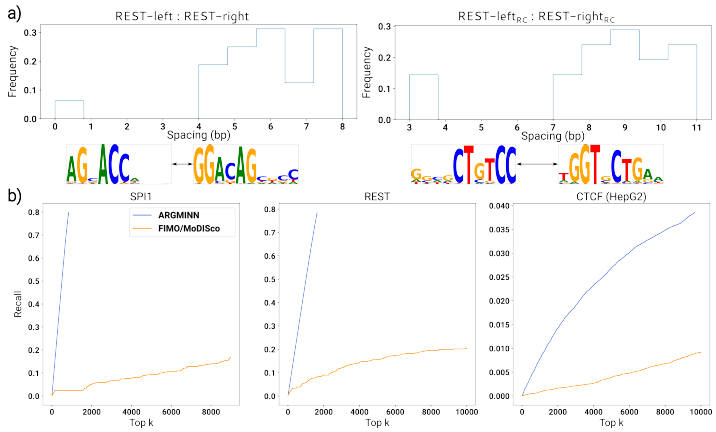

Figure 3: Motif instances and syntax. **a)** On our REST dataset, we recovered the binding pattern of the two halves of REST (both halves bind together, either adjacently or around 4–9 bp apart). We show the histogram of spacings between motif halves (spacings are in context of the 10 bp filters). **b)** We show the number of top k motif instances which were found to overlap the ground-truth motif or known DNA-binding footprints, comparing ARGMINN versus MoDISco followed by FIMO.

6 Robustness of ARGMINN

Because ARGMINN makes decisions based on biologically meaningful units (i.e. motifs), it is more robust to background variations. On our REST dataset, we trained ARGMINN and a traditional CNN on sequences with a 50% GC background, and then computed prediction accuracy on test sequences with GC content from 0% to 100%. ARGMINN's performance suffered less on different backgrounds (Supplementary Figure S4)

Furthermore, because traditional CNNs learn filters which do not accurately represent motifs, they are prone to adversarial attacks. On our SPI1 dataset, we trained a CNN to achieve 88% accuracy. We then easily constructed many sequences which contain short substrings which highly activate its filters, but without any instances of the SPI1 motif. On this set of sequences, the traditional CNN's accuracy dropped to 55%. Reflexively, we also were able to easily design sequences which have the SPI1 motif, but we inserted substrings into the background which strongly deactivate the filters, leading the CNN's accuracy to drop to 48%. In contrast, because ARGMINN encodes meaningful biological motifs in each filter, it remains robust against such an attack (i.e. one cannot easily identify highly-activating or anti-activating sequences which trick the model into giving the wrong prediction).

7 Discussion

We illustrated ARGMINN's unique ability to reveal motifs and their syntax directly from its weights and activations—an advantage which is entirely absent from traditional (non-mechanistically interpretable) DNNs. We then compared the predictive performance of ARGMINN to a traditional CNN (Supplementary Table S1). It is generally well known that more interpretable models may suffer slightly in predictive performance. This is expected, as these models tend to base decisions on human-interpretable concepts (e.g. crucial motifs), instead of spurious signals which can be informative, but are not useful for understanding regulatory genomics (e.g. GC content, exceptionally rare motifs, etc.). In our experiments, however, we found that ARGMINN achieved better or comparable performance to its corresponding, non-interpretable baseline. This demonstrates a further benefit of mechanistic interpretability, in that it may help a model achieve even better predictive performance by focusing on the most biologically meaningful signals (e.g. motifs).

Mechanistic interpretability is still a nascent field, and our research pioneers an architecture that enables direct interpretation with minimal post-hoc analysis. We showed that ARGMINN is expressive enough to accurately predict genome regulation, yet is uniquely constrained so that the weights and activations directly encode the model's decision process *in a human-interpretable way*. Further work in this area can yield major benefits for scientific AI and explainable AI.

REFERENCES

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 Biotechnology*, 33:831–838, 8 2015. ISSN 15461696. doi: 10.1038/nbt.3300.

David Alvarez-Melis and Tommi S. Jaakkola. On the robustness of interpretability methods. 6 2018. URL https://arxiv.org/abs/1806.08049v1.

Timothy L Bailey, James Johnson, Charles E Grant, and William S Noble. The meme suite. *Nucleic Acids Research*, 43:W39–49, 7 2015. doi: 10.1093/nar/gkv416.

Pietro Barbiero, Gabriele Ciravegna, Francesco Giannini, Mateo Espinosa Zarlenga, Lucie Charlotte Magister, Alberto Tonda, Pietro Lio, Frederic Precioso, Mateja Jamnik, and Giuseppe Marra. Interpretable neural-symbolic concept reasoning, 6 2023.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 5:e24, 3 2020. ISSN 2476-0757. doi: 10.23915/DISTILL.00024. URL https://distill.pub/2020/circuits.

Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, Adrià Garriga-Alonso, and † Independent. Towards automated circuit discovery for mechanistic interpretability. 4 2023. URL https://arxiv.org/abs/2304.14997v1.

ENCODE Project Consortium. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489:57–74, 9 2012. doi: 10.1038/nature11247.

Oriol Fornes, Jaime A Castro-Mondragon, Aziz Khan, Robin van der Lee, Xi Zhang, Phillip A Richmond, Bhavi P Modi, Solenne Correard, Marius Gheorghe, Damir Baranašić, Walter Santana-Garcia, Ge Tan, Jeanne Chèneby, Benoit Ballester, François Parcy, Albin Sandelin, Boris Lenhard, Wyeth W Wasserman, and Anthony Mathelier. Jaspar 2020: update of the open-access database of transcription factor binding profiles. *Nucleic Acids Research*, 48:D87–D92, 1 2020. doi: 10.1093/nar/gkz1001.

Ryan J. Geusz, Allen Wang, Dieter K. Lam, Nicholas K. Vinckier, Konstantinos Dionysios Alysandratos, David A. Roberts, Jinzhao Wang, Samy Kefalopoulou, Araceli Ramirez, Yunjiang Qiu, Joshua Chiou, Kyle J. Gaulton, Bing Ren, Darrell N. Kotton, and Maike Sander. Sequence logic at enhancers governs a dual mechanism of endodermal organ fate induction by foxa pioneer factors. *Nature Communications 2021 12:1*, 12:1–19, 11 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-26950-0. URL https://www.nature.com/articles/s41467-021-26950-0.

Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 3681–3688, 10 2017. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33013681. URL https://arxiv.org/abs/1710.10547v2.

Amirata Ghorbani, James Wexler Google Brain, James Zou, and Been Kim Google Brain. Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://github.com/amiratag/ACE.

Ehsan Hajiramezanali, Sepideh Maleki, Alex Tseng, Aicha BenTaieb, Gabriele Scalia, and Tommaso Biancalani. On the consistency of gnn explainability methods. In *XAI in Action: Past, Present, and Future Applications*, 2023.

David R. Kelley, Jasper Snoek, and John L. Rinn. Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, 26:990–999, 7 2016. ISSN 15495469. doi: 10.1101/gr.200535.115.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *35th International Conference on Machine Learning, ICML 2018*, 6:4186–4195, 11 2017. URL https://arxiv.org/abs/1711.11279v5.

Pang Wei Koh, Thao Nguye, Yew Siang Tang, Stephen Mussmann, Emma Pierso, Been Kim, and Percy Liang. Concept bottleneck models. *37th International Conference on Machine Learning, ICML 2020*, PartF168147-7:5294–5304, 7 2020. URL https://arxiv.org/abs/2007.04612v3.

Samuel A Lambert, Arttu Jolma, Laura F Campitelli, Pratyush K Das, Yimeng Yin, Mihai Albu, Xiaoting Chen, Jussi Taipale, Timothy R Hughes, and Matthew T Weirauch. The human transcription factors. *Cell*, 172:650–665, 2 2018. doi: 10.1016/j.cell.2018.01.029.

Xiao Liu, Jun Xu, Sara Rosenthal, Ling juan Zhang, Ryan McCubbin, Nairika Meshgin, Linshan Shang, Yukinori Koyama, Hsiao Yen Ma, Sonia Sharma, Sven Heinz, Chris K. Glass, Chris Benner, David A. Brenner, and Tatiana Kisseleva. Identification of lineage-specific transcription factors that prevent activation of hepatic stellate cells and promote fibrosis resolution. *Gastroenterology*, 158:1728–1744.e14, 5 2020. ISSN 0016-5085. doi: 10.1053/J.GASTRO.2020.01.027.

Ziming Liu, Eric Gan, and Max Tegmark. Seeing is believing: Brain-inspired modular training for mechanistic interpretability. 5 2023. URL https://arxiv.org/abs/2305.08746v2.

Gherman Novakovsky, Nick Dexter, Maxwell W. Libbrecht, Wyeth W. Wasserman, and Sara Mostafavi. Obtaining genetics insights from deep learning via explainable artificial intelligence. *Nature Reviews Genetics 2022 24:2*, 24:125–137, 10 2022. ISSN 1471-0064. doi: 10.1038/s41576-022-00532-2. URL https://www.nature.com/articles/s41576-022-00532-2.

Darcie D. Seachrist, Lindsey J. Anstine, and Ruth A. Keri. Foxa1: A pioneer of nuclear receptor action in breast cancer. *Cancers*, 13, 10 2021. ISSN 20726694. doi: 10.3390/CANCERS13205205. URL /pmc/articles/PMC8533709//pmc/articles/PMC8533709/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC8533709/.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *Proceedings of Machine Learning Research*, pp. 3145–3153, 7 2017. ISSN 1938-7228. URL http://proceedings.mlr.press/v70/shrikumar17a.html.

Avanti Shrikumar, Katherine Tian, Anna Shcherbina, Žiga Avsec, Abhimanyu Banerjee, Mahfuza Sharmin, Surag Nair, and Anshul Kundaje. Tf-modisco v0.4.2.2-alpha: Technical note. *arXiv*, 10 2018. URL http://arxiv.org/abs/1811.00416.

Trevor Siggers and Raluca Gordân. Protein–dna binding: complexities and multi-protein codes. *Nucleic Acids Research*, 42:2099–2111, 2 2014. ISSN 0305-1048. doi: 10.1093/NAR/GKT1112.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv*, 3 2017. URL http://arxiv.org/abs/1703.01365.

Alex M. Tseng, Avanti Shrikumar, and Anshul Kundaje. Fourier-transform-based attribution priors improve the interpretability and stability of deep learning models for genomics. *bioRxiv*, pp. 2020.06.11.147272, 6 2020. doi: 10.1101/2020.06.11.147272. URL https://www.biorxiv.org/content/10.1101/2020.06.11.147272v1https://www.biorxiv.org/content/10.1101/2020.06.11.147272v1.abstract.

Jeff Vierstra, John Lazar, Richard Sandstrom, Jessica Halow, Kristen Lee, Daniel Bates, Morgan Diegel, Douglas Dunn, Fidencio Neri, Eric Haugen, Eric Rynes, Alex Reynolds, Jemma Nelson, Audra Johnson, Mark Frerker, Michael Buckley, Rajinder Kaul, Wouter Meuleman, and John A. Stamatoyannopoulos. Global reference mapping of human transcription factor footprints. *Nature 2020 583:7818*, 583:729–736, 7 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2528-x. URL https://www.nature.com/articles/s41586-020-2528-x.

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 12:931–934, 9 2015. ISSN 15487105. doi: 10.1038/nmeth.3547.

Žiga Avsec, Melanie Weilert, Avanti Shrikumar, Sabrina Krueger, Amr Alexandari, Khyati Dalal, Robin Fropf, Charles McAnany, Julien Gagneur, Anshul Kundaje, and Julia Zeitlinger. Base-resolution models of transcription-factor binding reveal soft motif syntax. *Nature Genetics*, 53:354–366, 3 2021. doi: 10.1038/s41588-021-00782-6.

A SUPPLEMENTARY ALGORITHMS, FIGURES, AND TABLES

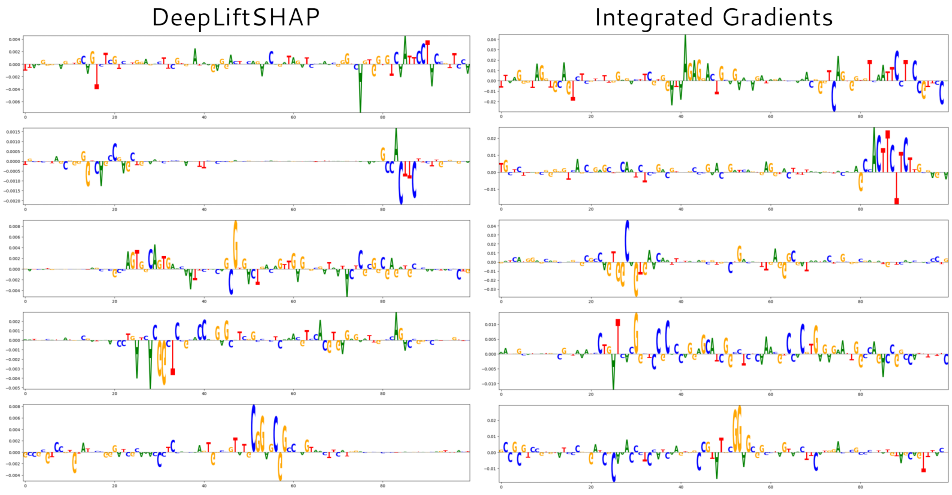

Figure S1: Examples of importance scores for SPI1. On our standard CNN trained on the simulated SPI1 dataset, we extracted importance scores from input sequences using two different methods: DeepLIFTShap and integrated gradients. We show the importance scores computed by each method for 5 randomly selected motif-containing sequences. Note that the importance scores are highly noisy, and regardless of the method, it remains difficult to even identify *where* the motif is from these score tracks, let alone *what* the motif is. Additionally, note the high degree of disagreement between the two methods, with each method often disagreeing on the sign of the importance for an important subsequence.

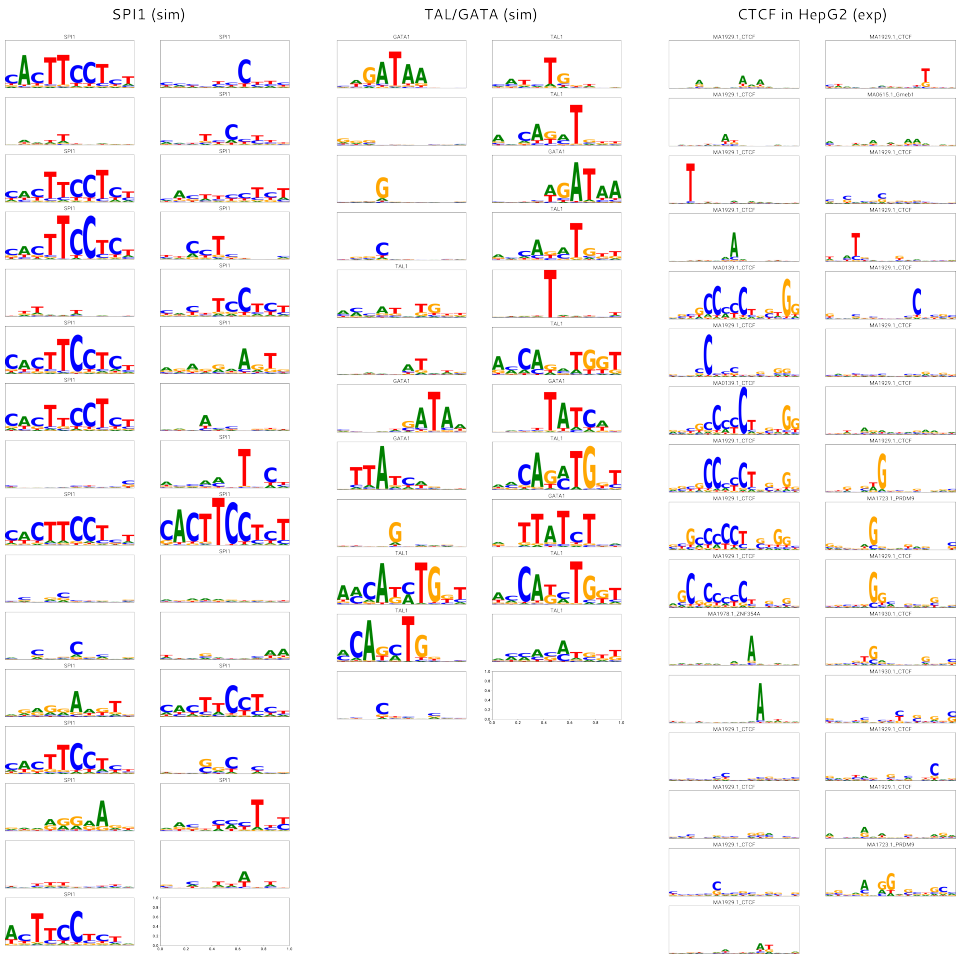

Figure S2: Motifs discovered with the MoDISco pipeline. For our simulated SPI1 and TAL/GATA datasets, and our experimental CTCF dataset, we computed importance scores and ran the MoDISco clustering algorithm to discover motifs. We show the discovered motifs which pass the internal MoDISco filters, ranked by number of supporting seqlets (i.e. the prevalence of the motif according to the MoDISco algorithm) in columns from left to right. Each motif is labeled with the closest matching possible motif in the dataset using TOMTOM. For simulated datasets, we matched each motif to the closest motif in the simulation; for the experimental dataset, we matched each motif to the closest known human motif. Note that MoDISco combines the forward and reverse-complement orientations of motifs. Unlabeled motifs are those where TOMTOM failed to find a match. Even after clustering, there is a large amount of redundancy between motifs (i.e. many motifs are learning the same patterns redundantly). Furthermore, there are many motifs discovered which are extraneous (i.e. not relevant or correct). These motifs tend to be very short and have low information content, and they also often have a *higher prevalence* than the relevant true motifs.

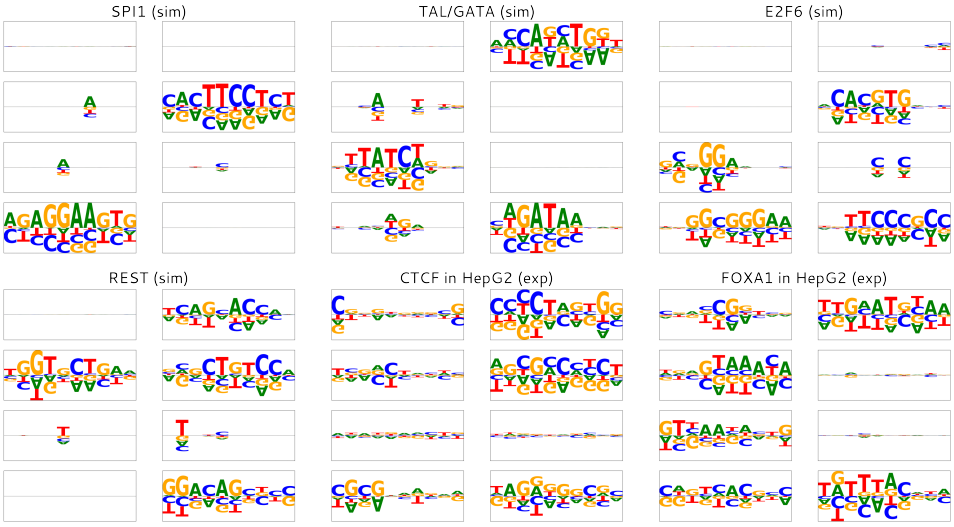

Figure S3: Learned filters from traditional CNN with filter regularization. We applied our filter regularizer (Equation 2) to the traditional CNN architecture, and show the resulting learned filter weights. With the filter regularization the traditional CNN now also is able to show clean, non-redundant, relevant motifs in its filters. Thus, our filter regularizer is able to turn even traditional networks into more mechanistically interpretable architectures. Importantly, although a traditional CNN with our filter regularizer can now reveal discovered motifs, without our unique attention mechanism (Equation 3), it still is unable to easily reveal motif instances and syntax. As such, these partially interpretable architectures would still need to rely on traditional motif-instance-scanning algorithms.

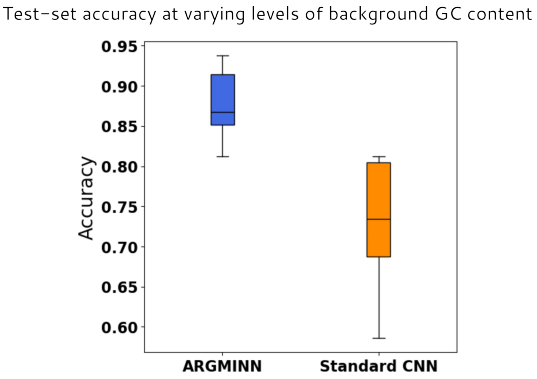

Figure S4: Example of robustness to sequence context. On our simulated REST dataset, we tested the ability of ARGMINN or a traditional CNN to predict accurately on sequences with GC content different from what they were trained with. We tested background sequences with GC content ranging from 10% to 90% (in intervals of 10%). The traditional CNN, in addition to achieving lower performance in general, shows a much larger range of performance.

Table S1: Test-set accuracy

	SPI1	TAL/GATA	E2F6	REST	CTCF (HepG2)	FOXA2 (HepG2)
ARGMINN	**89.8%**	**88.4%**	**90.8%**	**88.3%**	76.9%	72.8%
Traditional CNN	88.7%	83.8%	89.5%	77.1%	**77.5%**	**73.4%**

B SUPPLEMENTARY METHODS

We trained all of our models and performed all analyses on a single Nvidia Quadro P6000.

B.1 TRAINING DATA

For our simulated datasets, we downloaded motif PFMs (position frequency matrices) from JASPAR Fornes et al. (2020), and trimmed off low-information-content flanks. When training and testing, we randomly generated 500 bp sequences on the fly. Motif instances were sampled from the PFMs, and inserted into a randomly sampled background (from a uniform distribution of A, C, G, and T). Motifs (or combinations of them) were randomly inserted in the central 100 bp of the background. Our simulations contained motif configurations as follows:

- SPI1: all positive sequences have a single instance of the SPI1 motif
- TAL/GATA: 37.5% of positive sequences have a single instance of TAL1, 37.5% have a single instance of GATA1, and 25% of sequences have both (either 7, 8, or 9 bp apart)
- E2F6: 10% of positive sequences have a single instance of E2F6, and 90% have both E2F6 and MAX, between 30 and 60 bp apart; 50% of negative sequences have only the MAX motif, and the other 50% are random background
- REST: all positive sequences have both the left- and right-half motifs, with a spacing of 2, 6, 7, 8, 9, 10, or 11 bp apart; 25% of negative sequences have only the left-half motif, 25% have only the right-half motif, and 50% are random background

When generating simulated sequences, we scan the random backgrounds for spurious matches and filter out such instances. Note that this is a crucial step for simulated datasets, as for short motifs such as GATA (6 bp), we would expect over 12% of random backgrounds to contain a perfect match by chance. This is an issue which is much rarer in real datasets, but for our simulations, we scan the PFMs (and their reverse complements) across our generated backgrounds (for positive and negative sequences), and ensure that no PFM attains a match score of over 0.9.

For our experimental datasets, we downloaded the IDR peaks from ENCODE Consortium (2012) for experiments ENCSR607XFI (CTCF in HepG2) and ENCSR865RXA (FOXA1 in HepG2). Our positive dataset consisted of random 500 bp sequences drawn from the genome, where at least half of the 500 bp overlaps a peak (if the peak is less than 500 bp), or at least half of the peak overlaps the 500 bp (if the peak is over 500 bp). Our negative dataset consisted of randomly sampled intervals from the genome.

Note that when training, we automatically use reverse-complement augmentation so that every batch is doubled in size with its reverse complement.

B.2 MODEL ARCHITECTURES

Our ARGMINN architecture consists of two modules: motif scanners and a syntax builder. The motif scanner consists of a single convolutional layer of 8 filters, each of 10 bp in width, which scan across a one-hot-encoded DNA sequence. The result is passed to a ReLU, which constitutes the "motif activations".

The motif activations are concatenated with a positional encoding of dimension $d = 16$. Our positional encoding is defined as follows:

$$P_{i,2j} = \sin(\frac{i}{10000^{\frac{2i}{d}}}), P_{i,2j+1} = \cos(\frac{i}{10000^{\frac{2i}{d}}})$$

where i is the position along the sequence, and $j \in \{0, ..., \frac{d}{2} - 1\}$.

Let $A \| P$ be the motif activations and positional encodings concatenated together. This is passed to two consecutive memory-stream-based attention layers. The lth attention layer starts with a memory stream m_{l-1} of dimension 128 (m_0 is a vector of all 1s), and $A \| P$. In each layer, m_{l-1} is passed through a linear layer to obtain a single query q_l of dimension equal to the dimension of each input

token in $A \| P$. Two separate linear layers also convert each token in $A \| P$ into a set of key vectors and value vectors (of the same dimension as the input token). The query, keys, and values are reshaped to obtain 4 attention heads. We then compute the attention scores by multiplying the query against all keys, and normalizing by $\sqrt{d_{q_l}}$, where d_{q_l} is the dimension of the query/key/value vectors. We then perform dropout on the attention scores with dropout rate 0.1, and softmax the scores for each attention head. These attention scores are used to weight the value vectors in a weighted sum, which is then reshaped to reincorporate the heads. This is passed through another linear layer which retains the same dimension, followed by dropout and layer normalization. This is then fed to a 2-layer MLP with ReLU and dropout in between the two linear layers, mapping the result to the same dimension as m_{l-1}. After a final dropout and layer norm, this is added to m_{l-1} to obtain m_l.

After all attention layers, the final m_l is passed to a single linear layer which maps it to a scalar prediction which is passed to a sigmoid activation function (for binary prediction).

Our standard CNN follows an architecture which is common in the literature for single-task predictions. We apply 3 successive convolutional layers to the input sequence, each with 8 filters and of filter width 10, 5, and 5. After each layer we pass through a ReLU and apply batch normalization. We then perform max pooling with a filter size of 40 and a stride of 40. This is passed to two linear layers of 10 and 5 hidden dimensions each. After each linear layer, we pass through a ReLU and apply batch normalization. Finally, a final linear layer maps the result to an sigmoid-activated output.

B.3 Training schedules

When training, we used a batch size of 128 with an equal number of positives and negatives (this includes the reverse-complement augmentation). For our simulated datasets, each training epoch consisted of 100 batches, and each validation and test epoch consisted of 10 batches. For our experimental datasets, we reserved chr8 and chr10 for validation, and chr1 for test (all other chromosomes were used for training).

We trained all of our models for 40 epochs (regardless of architecture), and noted that the loss had converged in all cases. We used a learning rate of 0.001, and early stopping (requiring an improvement of at least 0.001 in the loss at least once over the past 3 epochs).

For ARGMINN, we weighted the secondary losses as follows:

- λ_o: 0 for the first 10 epochs, increasing from $10^{0.5}$ to 10^4 evenly in logarithmic space over the next 20 epochs, and stable at 10^4 for the last 10 epochs
- λ_l: 0 for the first 10 epochs, increasing from 10^{-4} to 10^{-3} evenly in logarithmic space over the next 20 epochs, and stable at 10^{-3} for the last 10 epochs

For all of our models, we train 3 random initializations and select the one with the best test accuracy for downstream analyses.

B.4 Analyses

Extracting motifs from convolutional filters

To extract motifs from convolutional filters, we simply visualized the multiplicative weights of each filter as a sequence logo. We also applied mean centering at each position independently.

For motifs deriving from convolutional filters, we convert them to PFMs by computing the average of sequences (assuming a uniformly random background) which activate that filter to at least 50% of its maximum possible activation. This is computed empirically over 1 million sequences. For motifs from MoDISco, we simply took the PFM output and cut off low-information-content flanks until it was the same size as the filters.

To compute matches to known motifs, we used TOMTOM (Bailey et al., 2015) to compute the q-value similarity between a PFM to known motifs (across all possible alignments). We report the $-\log_{10}(q)$ value as similarity. For simulated datasets, we reported the closest match (i.e. highest similarity) to any motif in the dataset. For experimental datasets, we reported the closest match to any relevant motif in the JASPAR human-motif database Fornes et al. (2020). The set of relevant motifs for experimental datasets was decided by first running TOMTOM against all known human

motifs and pooling together the top matches (by motif family) over all methods and architectures (e.g. ARGMINN, MoDISco, etc.). Relevant families were identified as those with support from existing literature.

Tracing back motif instances and syntax

To trace back motif instances for a particular input sequence in ARGMINN, we performed a forward pass and retained the motif activations and attention scores. Over all layers and all attention heads, we examined the positions in the sequence which had an attention score of at least 0.9. We then called a motif hit if the activation for a filter at that position was at least the average activation over the entire batch (over all positions in the sequences).

To identify syntax, we separated the input sequences by which motif instances were called, and computed the distribution of the spacings between the motif instances.

In order to rank motif instances from ARGMINN, we ranked by maxmimum attention score over all heads/layers at that position. To break any ties, we used the highest motif-filter activation score at that position.

Computing importance scores

We computed importance scores using PyTorch Captum, with DeepLIFTShap or with integrated gradients Shrikumar et al. (2017); Sundararajan et al. (2017). For DeepLIFTShap, we used a reference of 10 baselines consisting of dinucleotide-shuffled sequences, as recommended in Shrikumar et al. (2017), recovering the hypothetical importance scores at each position. For integrated gradients, we used a baseline of all 0s.

Discovering motifs with MoDISco

We computed importance scores over 1280 positive input sequences using the DeepLIFTShap algorithm as described above. We then ran MoDISco-lite v2.2.0 Shrikumar et al. (2018) using a maximum of 10000 seqlets and default parameters.

Scanning for motif instances with FIMO

To scan for motif instances using FIMO, we started with PFMs and ran FIMO v5.0.5 Bailey et al. (2015) on a batch of 128 positive sequences and their reverse complements. We used the default FIMO parameters.

To rank FIMO hits, we used the q-value from FIMO. We also collapsed overlapping FIMO hits before analyzing.

Experimental ground-truth motif instances

For our experimental datasets, we obtained "ground-truth" motif instances from Vierstra et al. (2020), an independently collected set of DNA-binding footprints in various cell types. For each cell type of interest (e.g. HepG2), we simply pooled together the footprints of all experiments from that cell type and used those as a set of ground-truth motif instances.

Generating adversarial examples

We generated adversarial examples in a traditional CNN in two ways.

First, we generated sequences which had no motifs present, but were still predicted to have a positive label by the DNN. To do this, we built up random sequences by taking highly-activating sequences from random filters and adding them to the sequence. We took the most highly-activating sequence for each filter and randomly strung together a random ordering of such sequences to obtain a 500 bp example. We verified that no part of these sequences matched any motif (or reverse complement) in the dataset.

Next, we generated sequences which had motifs, but were still predicted to be negative by the DNN. To do this, we first sampled a motif configuration from the normal simulation. Under normal circumstances, such a configuration would endow a positive label in the dataset. We then surrounded either side of this configuration with sequences that are least activating for a random ordering of filters. We randomly strung together a random ordering of such sequences to pad out a 500 bp example.