Appendix for Learning Important Features Through Propagating Activation Differences

1 A: Proof of summation-to-delta with the multiplier chain rule

The chain rule for multipliers is:

$$m_{\Delta x_i \Delta z} = \sum_j m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \tag{1}$$

Where multipliers are defined as:

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \tag{2}$$

Given $C_{\Delta x_i \Delta y_j}$ and $C_{\Delta y_j \Delta z}$ both satisfy summation-to-delta, we show that defining $m_{\Delta x_i \Delta z}$ according to the chain rule also satisfies summation-to-delta (that is, $\sum_i C_{\Delta x_i \Delta z} = \Delta z$). We have:

$$\begin{aligned}
\sum_i C_{\Delta x_i \Delta z} &= \sum_i \Delta x_i m_{\Delta x_i \Delta z} \text{ (By definition of } m_{\Delta x_i \Delta z}) \\
&= \sum_i \Delta x_i \sum_j m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \text{ (By the chain rule)} \\
&= \sum_i \Delta x_i \sum_j \frac{C_{\Delta x_i \Delta y_j}}{\Delta x_i} m_{\Delta y_j \Delta z} \text{ (By definition of } m_{\Delta x_i \Delta y_j}) \\
&= \sum_i \sum_j C_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \\
&= \sum_j \sum_i C_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta z} \text{ (Flipping the order of summation)} \\
&= \sum_j \Delta y_j m_{\Delta y_j \Delta z} \text{ (By summation-to-delta of } C_{\Delta x_i \Delta y_j}) \\
&= \sum_j \Delta y_j \frac{C_{\Delta y_j \Delta z}}{\Delta y_j} \text{ (By definition of } m_{\Delta y_j \Delta z}) \\
&= \sum_j C_{\Delta y_j \Delta z} = \Delta z \text{ (By summation-to-delta of } C_{\Delta y_j \Delta z})
\end{aligned}$$

2 B: Propagation of Linear rule multipliers using standard neural network operations

We can frame the propagation of the multipliers for the Linear rule in terms of standard operations provided by GPU backends such as tensorflow and theano.

2.1 Dense layers

Let W represent the tensor of weights, and let ΔX and ΔY represent a 2d matrix with dimensions sample $\times$ features such that $\Delta Y = \text{matrix\_mul}(W, \Delta X)$. Here, matrix_mul is matrix multiplication. Let $M_{\Delta X \Delta t}$ and $M_{\Delta Y \Delta t}$ represent tensors of multipliers (again with dimensions sample $\times$ features). Let $\cdot$ represent an elementwise product, and let $1\{\text{condition}\}$ represent a binary matrix that is 1 where "condition" is true and 0 otherwise. We have:

$$\begin{aligned}
M_{\Delta X \Delta t} =& (\text{matrix_mul}(W^T \odot 1\{W^T > 0\}, M_{\Delta Y^+ \Delta t}) \odot 1\{\Delta X > 0\} \\
&+ \text{matrix_mul}(W^T \odot 1\{W^T < 0\}, M_{\Delta Y^+ \Delta t}) \odot 1\{\Delta X < 0\}) \\
&+ (\text{matrix_mul}(W^T \odot 1\{W^T > 0\}, M_{\Delta Y^- \Delta t}) \odot 1\{\Delta X < 0\} \\
&+ \text{matrix_mul}(W^T \odot 1\{W^T < 0\}, M_{\Delta Y^- \Delta t}) \odot 1\{\Delta X > 0\}) \\
&+ \text{matrix_mul}(W^T, 0.5(M_{\Delta Y^+ \Delta t} + M_{\Delta Y^- \Delta t})) \odot 1\{\Delta X = 0\})
\end{aligned}$$

2.2 Convolutional layers

Let W represent a tensor of convolutional weights such that $\Delta Y = \text{conv}(W, \Delta X)$, where conv represents the convolution operation. Let transposed_conv represent a transposed convolution (comparable to the gradient operation for a convolution) such that $\frac{d}{dt}X = \text{transposed\_conv}(W, \frac{d}{dt}Y)$. We have:

$$\begin{aligned}
M_{\Delta X \Delta t} =& (\text{transposed_conv}(W \odot 1\{W > 0\}, M_{\Delta Y^+ \Delta t}) \odot 1\{\Delta X > 0\} \\
&+ \text{transposed_conv}(W \odot 1\{W < 0\}, M_{\Delta Y^+ \Delta t}) \odot 1\{\Delta X < 0\}) \\
&+ (\text{transposed_conv}(W \odot 1\{W > 0\}, M_{\Delta Y^- \Delta t}) \odot 1\{\Delta X < 0\} \\
&+ \text{transposed_conv}(W \odot 1\{W < 0\}, M_{\Delta Y^- \Delta t}) \odot 1\{\Delta X > 0\}) \\
&+ \text{transposed_conv}(W, 0.5(M_{\Delta Y^+ \Delta t} + M_{\Delta Y^- \Delta t})) \odot 1\{\Delta X = 0\})
\end{aligned}$$

3 C: Detailed calculation of contributions in Fig. 3 using various backpropagation-based approaches

We reproduce Figure 3 from the main text below:

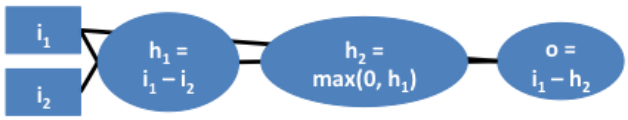

Figure 1: **Network computing** $o = \min(i_1, i_2)$**.**

We walk through the calculation of importance using various backpropagation methods. All methods except RevealCancel put importance either exclusively on i_1 or on i_2.

3.1 Gradients, Gradient $\times$ input and integrated gradients

Because $o = \min(i_1, i_2)$, we have $\frac{do}{di_1} = 0$ when $i_1 > i_2$ and $\frac{do}{di_2} = 0$ when $i_2 > i_1$. Thus, importance would be assigned either exclusively to i_1 or exclusively to i_2. Because the integrated gradients methods computes gradients at points interpolated linearly between 0 and the actual input values, it would also give importance either exclusively to i_1 or i_2.

3.2 Guided Backprop

We note that $\frac{do}{dh_2} < 0$. Thus, Guided Backprop would propagate zero importance to h_2, and all importance would be placed on i_1.

3.3 DeepLIFT with the Rescale rule

We compute the contributions and multipliers at every step of the calculation, assuming $i_1^0 = i_2^0 = 0$. First, we compute the reference activations for all the neurons by forward propagating the reference input, which gives: $h_1^0 = h_2^0 = o^0 = 0$. Then, we find the differences-from-reference, which are: $\Delta i_1 = i_1$, $\Delta i_2 = i_2$, $\Delta h_1 = h_1 = i_1 - i_2$, $\Delta h_2 = h_2 = \max(0, h_1)$ and $\Delta o = o = i_1 - h_2$.

$$
\begin{aligned}
C_{\Delta h_2 \Delta o} &= -\max(0, i_1 - i_2) \\
m_{\Delta h_2 \Delta o} &= -1 \\
C_{\Delta h_1 \Delta h_2} &= \max(0, i_1 - i_2) \\
m_{\Delta h_1 \Delta h_2} &= \frac{C_{\Delta h_1 \Delta h_2}}{\Delta h_1} = \frac{\max(0, i_1 - i_2)}{i_1 - i_2} \\
m_{\Delta h_1 \Delta o} &= m_{\Delta h_1 \Delta h_2} m_{\Delta h_2 \Delta o} = -\frac{\max(0, i_1 - i_2)}{i_1 - i_2} \\
C_{\Delta i_1 \Delta h_1} &= i_1 \\
m_{\Delta i_1 \Delta h_1} &= \frac{C_{\Delta i_1 \Delta h_1}}{\Delta i_1} = 1 \\
m_{\Delta i_1 \Delta o} &= 1 + m_{\Delta i_1 \Delta h_1} m_{\Delta h_1 \Delta o} = 1 - \frac{\max(0, i_1 - i_2)}{i_1 - i_2} \\
C_{\Delta i_2 \Delta h_1} &= -i_2 \\
m_{\Delta i_2 \Delta h_1} &= \frac{C_{\Delta i_2 \Delta h_1}}{\Delta i_2} = -1 \\
m_{\Delta i_2 \Delta o} &= m_{\Delta i_2 \Delta h_1} m_{\Delta h_1 \Delta o} = \frac{\max(0, i_1 - i_2)}{i_1 - i_2} \\
C_{\Delta i_1 \Delta o} &= \Delta i_1 m_{\Delta i_1 \Delta o} = i_1 \left(1 - \frac{\max(0, i_1 - i_2)}{i_1 - i_2}\right)
\end{aligned}
$$

$$C_{\Delta i_2 \Delta o} = \Delta i_2 m_{\Delta i_2 \Delta o} = i_2 \frac{\max(0, i_1 - i_2)}{i_1 - i_2}$$

Based on the formula above, we can see that when $i_1 - i_2 > 0$, we get $\frac{\max(0,i_1-i_2)}{i_1-i_2} = 1$, giving $C_{\Delta i_1 \Delta o} = 0$ and $C_{\Delta i_2 \Delta o} = i_2$. When $i_1 - i_2 < 0$, we have $C_{\Delta i_1 \Delta o} = i_1$ and $C_{\Delta i_2 \Delta o} = 0$.

3.4 DeepLIFT with the RevealCancel rule

We again start with $i_1^0 = i_2^0 = 0$. This gives $\Delta h_1^+ = i_1$, $\Delta h_1^- = -i_2$, $\Delta h_2 = \max(0, i_1 - i_2)$, $\Delta o^+ = i_1$ and $\Delta o^- = -h_2$. We compute Δh_2^+ and Δh_2^- as follows:

$$\begin{aligned}
\Delta h_2^+ &= \frac{1}{2}(\max(0, \Delta h_1^+) - \max(0, 0)) + \frac{1}{2}(\max(0, \Delta h_1^- + \Delta h_1^+) - \max(0, \Delta h_1^-)) \\
&= \frac{1}{2}(i_1 + \max(0, i_1 - i_2)) \\
\Delta h_2^- &= \frac{1}{2}(\max(0, \Delta h_1^-) - \max(0, 0)) + \frac{1}{2}(\max(0, \Delta h_1^+ + \Delta h_1^-) - \max(0, \Delta h_1^+)) \\
&= \frac{1}{2}(\max(0, i_1 - i_2) - i_1)
\end{aligned}$$

We then compute contributions as follows:

$$\begin{aligned}
C_{\Delta h_2 \Delta o} = C_{\Delta h_2 \Delta o^+} + C_{\Delta h_2 \Delta o^-} &= 0 + -h_2 = -\max(0, i_1 - i_2) \\
m_{\Delta h_2 \Delta o} = m_{\Delta h_2^+ \Delta o} = m_{\Delta h_2^- \Delta o} &= \frac{C_{\Delta h_2 \Delta o}}{\Delta h_2} = -1 \\
C_{\Delta h_1^+ \Delta h_2^+} &= \Delta h_2^+ \\
m_{\Delta h_1^+ \Delta h_2^+} &= \frac{\Delta h_2^+}{\Delta h_1^+} = \frac{0.5(i_1 + \max(0, i_1 - i_2))}{i_1} \\
m_{\Delta h_1^+ \Delta o} &= m_{\Delta h_1^+ \Delta h_2^+} m_{\Delta h_2^+ \Delta o} = \frac{-0.5(i_1 + \max(0, i_1 - i_2))}{i_1} \\
C_{\Delta h_1^- \Delta h_2^-} &= \Delta h_2^- \\
m_{\Delta h_1^- \Delta h_2^-} &= \frac{\Delta h_2^-}{\Delta h_1^-} = \frac{0.5(i_1 - \max(0, i_1 - i_2))}{i_2} \\
m_{\Delta h_1^- \Delta o} &= m_{\Delta h_1^- \Delta h_2^-} m_{\Delta h_2^- \Delta o} = \frac{-0.5(i_1 - \max(0, i_1 - i_2))}{i_2} \\
C_{\Delta i_1 \Delta h_1^+} &= i_1 \,;\, m_{\Delta i_1 \Delta h_1^+} = 1 \\
C_{\Delta i_1 \Delta h_1^-} &= 0 \,;\, m_{\Delta i_1 \Delta h_1^-} = 0 \\
C_{\Delta i_2 \Delta h_1^+} &= 0 \,;\, m_{\Delta i_2 \Delta h_1^+} = 0 \\
C_{\Delta i_2 \Delta h_1^-} &= -i_2 \,;\, m_{\Delta i_2 \Delta h_1^-} = -1 \\
m_{i_1 o} = 1 + m_{\Delta i_1 \Delta h_1^+} m_{\Delta h_1^+ \Delta o} + m_{\Delta i_1 \Delta h_1^-} m_{\Delta h_1^- \Delta o} &= 1 - \frac{0.5(i_1 + \max(0, i_1 - i_2))}{i_1} + 0 \\
m_{i_2 o} = m_{\Delta i_2 \Delta h_1^+} m_{\Delta h_1^+ \Delta o} + m_{\Delta i_2 \Delta h_1^-} m_{\Delta h_1^- \Delta o} &= 0 + \frac{0.5(i_1 - \max(0, i_1 - i_2))}{i_2} \\
C_{i_1 o} = \Delta i_1 m_{i_1 o} &= i_1 - 0.5(i_1 - \max(0, i_1 - i_2)) = 0.5(i_1 + \max(0, i_1 - i_2)) \\
C_{i_2 o} = \Delta i_2 m_{i_2 o} &= 0.5(i_1 - \max(0, i_1 - i_2))
\end{aligned}$$

In short, using the RevealCancel rule, we get that the contribution scores on both i_1 and i_2 are $0.5(i_1 - \max(0, i_1 - i_2))$. When $i_1 > i_2$, we have a contribution of $0.5i_2$ on both inputs, and when $i_1 < i_2$, we get a contribution of $0.5i_1$ on both inputs.

4 D: Architecture and training details for MNIST

We describe the architecture and training procedure for the MNIST model. This architecture was based on the MNIST architecture in the Keras examples, with strided convolutions replacing pooling layers and adjustments to the kernel size to cover all inputs due to the valid convolutions. The layers are as follows:

- Convolutional layer with 32 filters of kernel size (4,4) and stride (2,2)
- ReLU activation
- Convolutional layer with 64 filters of kernel size (4,4) and stride (2,2)
- ReLU activation
- Dropout with probability 0.25 of leaving out units
- Fully connected layer with 128 neurons
- ReLU activation
- Dropout with probability 0.5 of leaving out units
- Fully connected layer with 10 units
- Softmax output

The model was trained with a loss of categorical cross-entropy and the Adam optimizer with learning rate 0.001. We used a batch size of 128 and a training set consisting of 60K images. The testing set had 10K images. The model was trained for 19 epochs; training was terminated using early stopping. The final testing accuracy was 99.24%

5 E: Performance of MNIST without RevealCancel on all layers

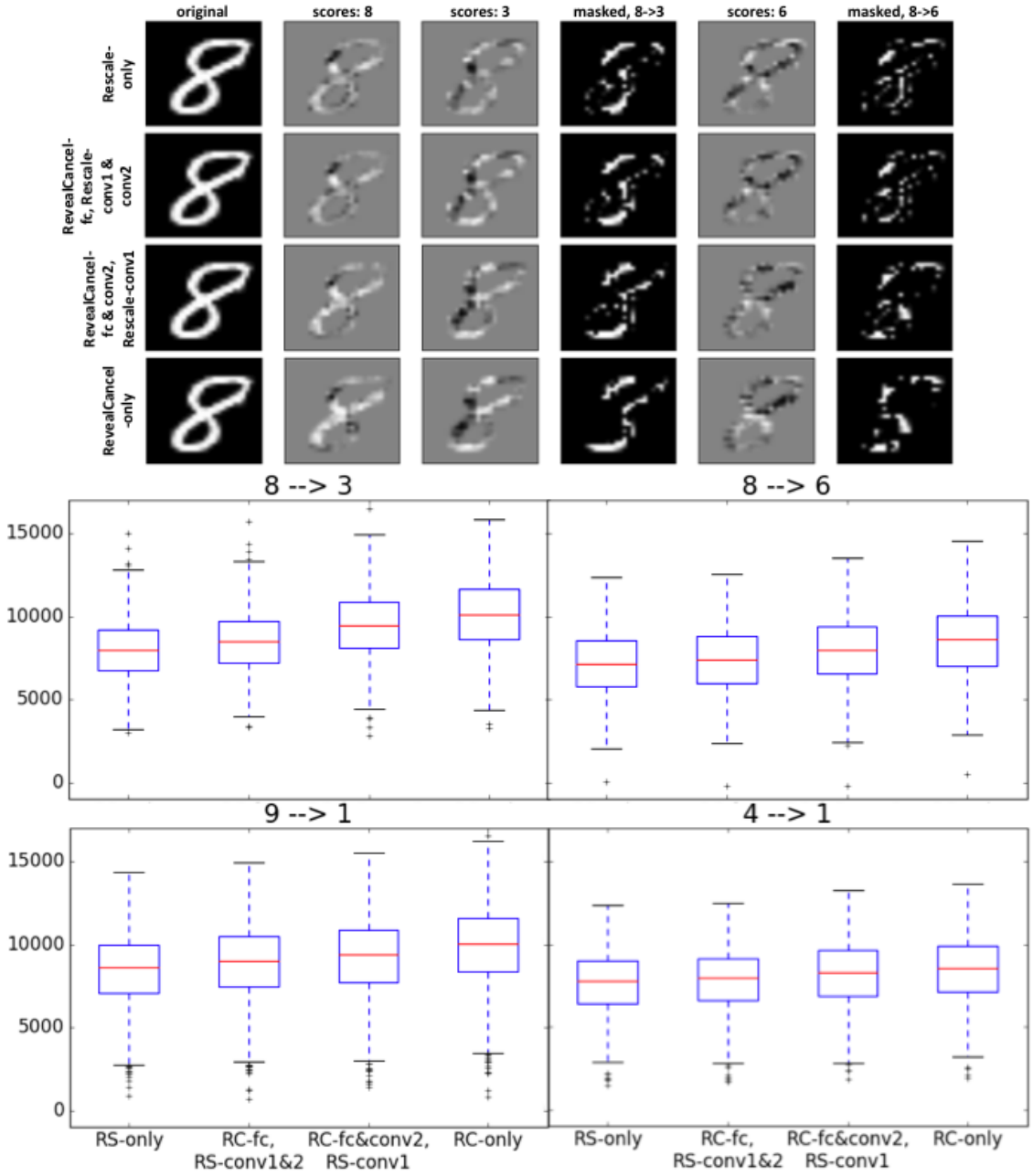

Figure 2: **RevealCancel applied to all layers outperforms RevealCancel applied only to some layers on MNIST.** The Y-axis represents the change in the log-odds score between the original class and the target class after pixel erasure (see **Fig. 4** in the main text for more details). We compared the performance of DeepLIFT with the RevealCancel used on all layers (RC-only), to the performance when the Rescale rule was used on the first convolutional layer instead of RevealCancel (RC-fc&conv2, RS-conv1), to the performance when the Rescale rule was used on the first two convolutional layers (RC-fc, RS-conv1&2), to the performance when only the Rescale rule was used (RS-only). Using the Rescale rule appeared to degrade performance for MNIST.

6 F: Details of genomics simulation and model architecture

6.1 Simulation details

The human genome has millions of DNA sequence elements (roughly 1000 letters long) containing specific combinations of short functional words to which regulatory proteins (RPs) bind to regulate gene activity. Each RP has binding affinity to specific collections of short DNA words (motifs). E.g. the GATA1 RP has high affinity to GATAA and moderate affinity to GATTA. Motifs of RPs are commonly represented as position weight matrix (PWMs) which are probability distributions over {A,C,G,T} at each position in the motif i.e. PWMs for a motif of length L is a matrix of probabilities of size 4 x L where the rows represent the 4 letters and the columns represent each position in the motif. The probabilities in each column sum to 1. PWMs are visualized as sequence logos (See `https://en.wikipedia.org/wiki/Sequence_logo` for additional details).

A key problem in computational genomics is the discovery of functional motifs in regulatory DNA sequence elements that give rise to distinct molecular signatures which can be measured experimentally (thus providing labels). Here, in order to benchmark DeepLIFT and other methods, we design a simulation that captures the essence of the motif discovery problem described above.

8K sequences of length 200 were simulated with a train-valid-test split of 0.8-0.1-0.1. First, a background sequence was generated by sampling the letters ACGT at each position with probabilities $0.3, 0.2, 0.2$ and 0.3 respectively. Then, motifs were sampled from PWMs [1] for the GATA1 and TAL1 regulatory proteins and inserted into the background sequence at random non-overlapping positions. For the GATA1 motif, we used the GATA_disc1 PWM and for the TAL1 motif, we used the TAL1_known1 PWM. $1/4$ of the sequences had both TAL1 and GATA1 motifs, $1/4$ had only TAL1 motifs, $1/4$ had only GATA1 motifs, and $1/4$ had no motifs embedded.

The number of each kind of motif to be inserted into a given sequence was determined by sampling from a Poisson distribution that was truncated to have a minimum of 1 and a maximum of 3 (when a number was sampled outside the range, it was resampled until a number within the range was achieved). For sequences containing either only GATA1 or only TAL1, the mean of the Poisson distribution was 2, and for sequences containing both GATA1 and TAL1, the mean of the Poisson for each motif was 1.

6.2 Architecture details

The input sequences were represented using one-hot encoding. We trained a model with the the following layers:

- Convolution with 50 filters of width 11
- ReLU
- Convolution with 50 filters of width 11
- ReLU
- Global Average Pooling
- Fully connected layer with 50 neurons
- ReLU
- Dropout layer with probability 0.5 of leaving out units
- Fully connected layer with 3 neurons
- Sigmoid output

The model was trained for 18 epochs with a binary crossentropy loss and the Adam optimizer and a learning rate of 0.001 (training was terminated with early stopping). The final auROC was 0.985 on Task 0, 0.983 on Task 1 and 0.997 on Task 2.

7 G: Equivalent of Fig. 5 in the main text for the GATA1 motif

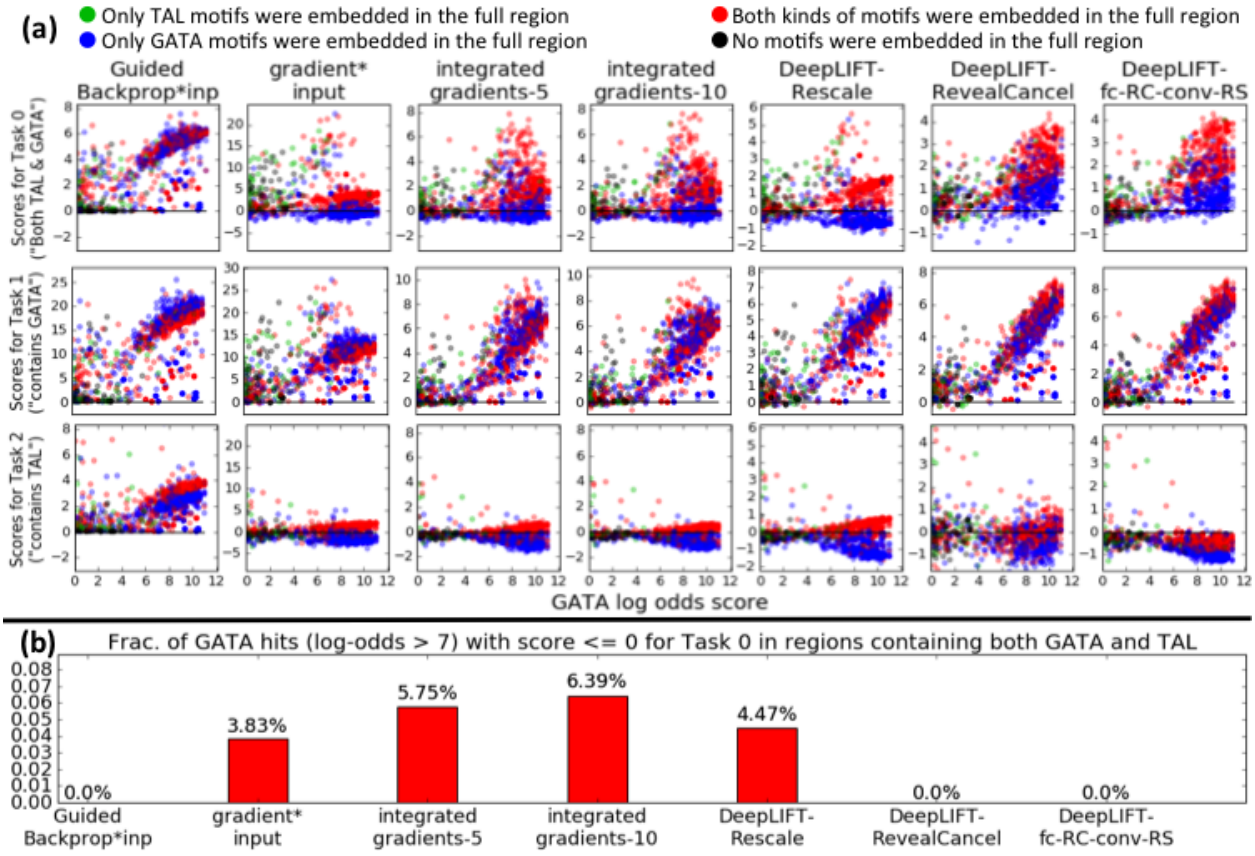

Figure 3: **DeepLIFT with RevealCancel gives qualitatively desirable behavior on TAL-GATA simulation**. (a) Scatter plots showing importance score vs. strength of GATA1 motif match for different tasks and methods. For each region, the top 5 motif matches are plotted. X-axes: log-odds score of GATA1 motif vs. background. Y-axes: total importance assigned to the match for the specified task. Red dots are from regions where both TAL1 and GATA1 motifs were inserted during simulation; blue is GATA1-only, red is TAL1-only, black is no motifs inserted. "DeepLIFT-fc-RC-conv-RS" refers to using the RevealCancel rule on the fully-connected layer and the Rescale rule on the convolutional layers, which appears to reduce noise relative to using RevealCancel on all the layers. (b) proportion of strong matches (log-odds > 7) to GATA1 motif in regions containing both TAL1 and GATA1 that had total score ≤ 0 for task 0; Guided Backprop$\times$inp and DeepLIFT with RevealCancel have no false negatives, but Guided Backprop has false positives for Task 2 (see Panel (a)).

8 H: Non-specific firing of Guided Backprop example sequence

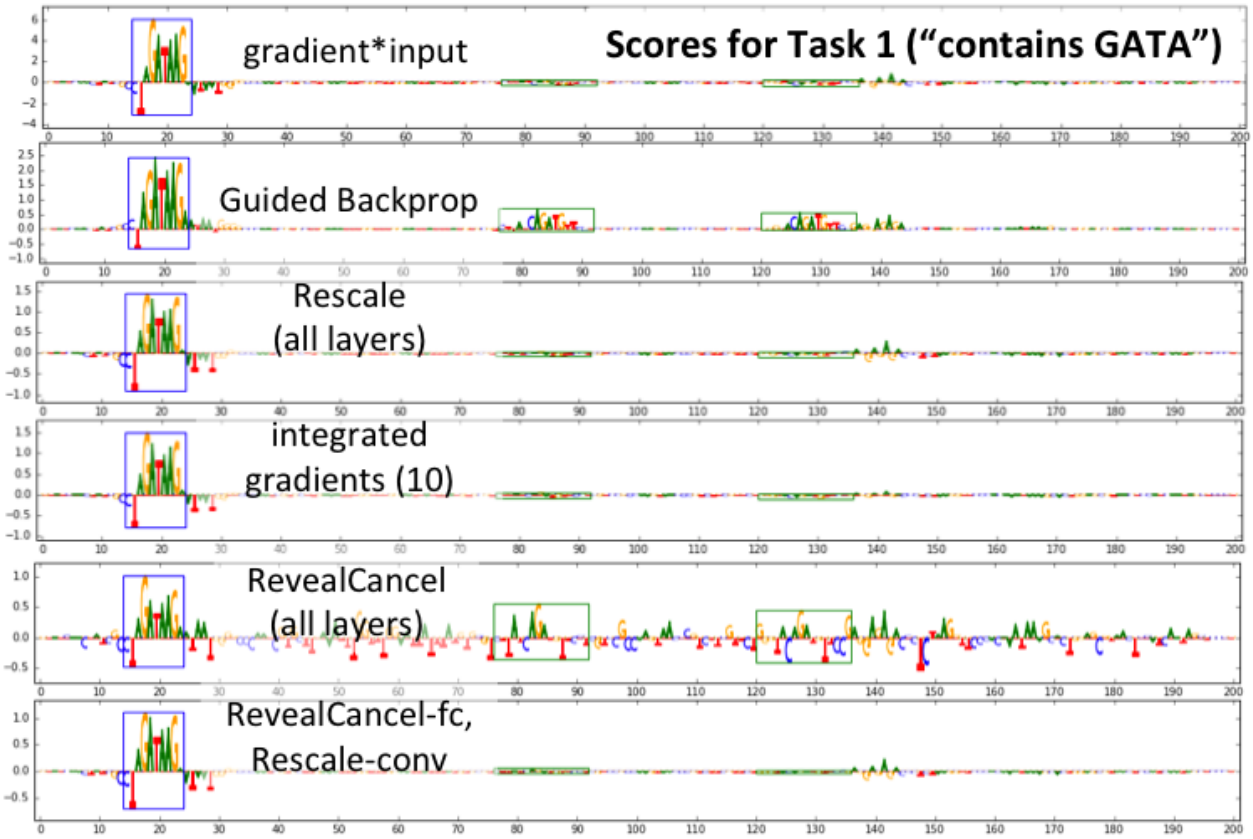

Figure 4: **Guided Backprop×input can lead to non-specific highlighting of sequence features**. Scores for Task 1 ("contains GATA1") computed using various methods. Heights of letters indicate sizes of importance scores. Blue boxes indicate locations of embedded GATA1 motifs and green boxes indicate locations of embedded TAL1 motifs. Guided Backprop×input (simply annotated as "Guided Backprop" in the figure) highlights the TAL1 motif as being relevant for the "contains GATA1" task.

9 I: Tiers in importance score are caused by having multiple instances of motifs

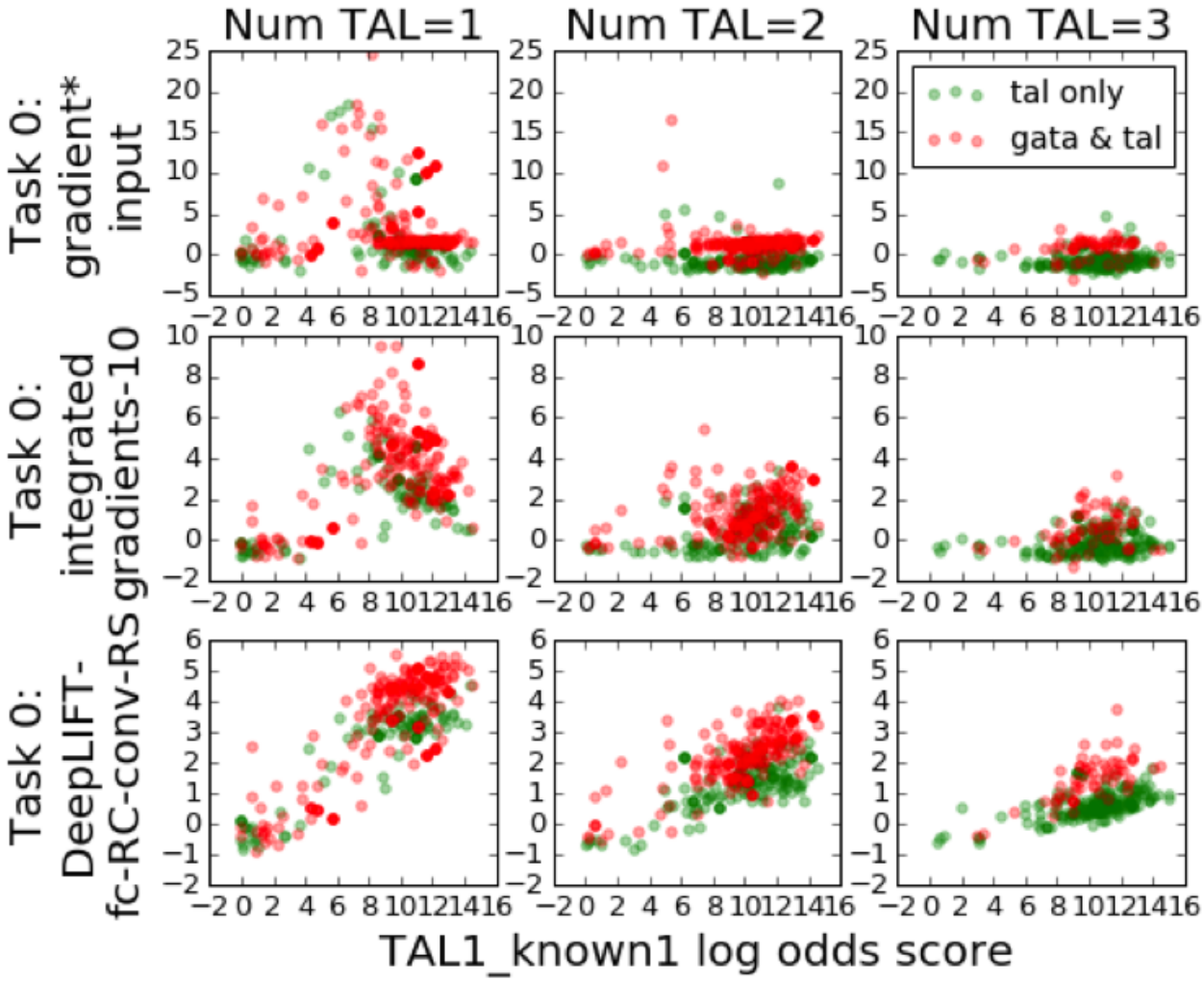

Figure 5: **Magnitude of scores on individual motif instances for Task 0 ("Both TAL1 and GATA1") depends on total number of motif instances in the sequence.** Top 5 matches (as ranked by the log-odds score) to the TAL1 motif for each sequence in the test set are plotted. X-axis indicates strength of the log-odds score of the match, and y-axis indicates total importance score for Task 0 assigned to match. Plots are separated by whether the entire sequence contains 1, 2 or 3 instances of the TAL1 motif. As illustrated, for sequences that contain multiple instances of the TAL1 motif, integrated gradients and DeepLIFT assign less importance to each individual instance of the motif. In each case, TAL1 motifs from sequences that contain a GATA1 motif (red dots) tend to receive more importance that TAL1 motifs from sequences that do not contain a GATA1 motif (green dots), consistent with property (5) discussed in Section 4.2.

10 J: Results with shuffled sequences as a reference

One undesirable aspect of using the frequencies of ACGT as the reference is that such a representation is not a one-hot encoding and thus never occurs in the training data. Thus, the network may behave unexpectedly on the reference sequence. As an alternative, we can average results over several one-hot encoded sequences generated by shuffling the original sequence. The results are shown below. On real genomic data, a dinucleotide-preserving shuffle would be more appropriate than a random shuffle as CG dinucleotides are underrepresented in genomic sequence due to spontaneous deamination from C to T.

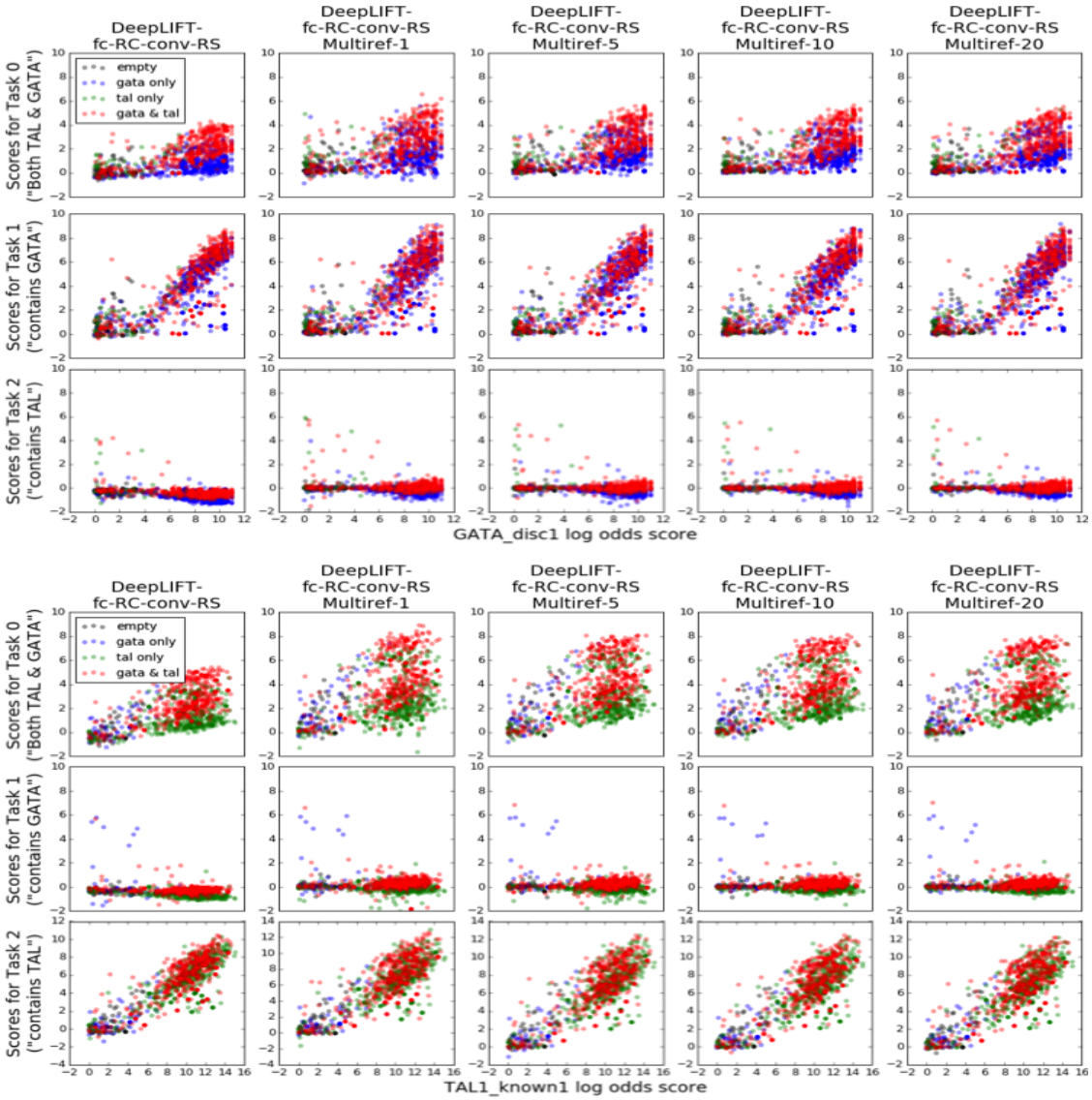

Figure 6: **Shuffled sequences are a viable choice of reference for genomic data**. For each sequence, scores were computed by averaging results over multiple references generated by randomly shuffling the original sequence. First column has results from using ACGT frequency as a reference; remaining columns use shuffled one-hot encoded sequences as reference. Multiref-n means n references were generated per sequence. Scores were computed using the RevealCancel rule on the fully-connected layer and the rescale rule on the convolutional layers. Shuffled sequences seem to result in higher DeepLIFT scores relative to ACGT frequency as the reference, and tiers caused by multiple motif instances are more noticeable (see **Appendix J**). Other qualitative aspects appear to be similar.

11 K: Weight normalization for one-hot encoded inputs

While it may seem natural to use a reference input of all zeros for one-hot encoded inputs, this neglects the fact that one-hot encoded inputs have the constraint that one of the inputs will always be a 1. For genomics, we suggest using a reference that is the average of one-hot encoded sequences from the negative set. However, if one would prefer to use a reference of all zeros, we propose applying the following output-preserving transformation that guarantees the activation of a convolutional filter given an input of all-zeros is equal to the average activation over all possible one-hot encoded inputs:

Let W and b represent the weights and bias of a 1d convolutional filter such that for a given input patch X, the output y of the convolutional neuron is:

$$y = \sum_i \sum_j W_{ij} X_{ij} + b \tag{3}$$

Here, i iterates over the length dimension and j iterates over the channel dimension. We also assume that the inputs are one-hot encoded, that is:

$$\sum_j X_{ij} = 1 \tag{4}$$

Let n be the number of channels. We propose the following transformation:

$$W'_{ij} = W_{ij} - \frac{1}{n} \sum_j W_{ij} \tag{5}$$

$$b' = b + \frac{1}{n} \sum_i \sum_j W_{ij} \tag{6}$$

In other words, we normalize the weights at each position by subtracting the mean weight across all channels, and add the mean to the bias to preserve the output. Because the weights are now mean-normalized, the average output over all possible one-hot encoded values for X is equal to the output when X is zero everywhere (which in turn is equal to the new bias b'). Doing this normalization can improve results when using a reference of all-zeros.

We can prove that the output is preserved on one-hot encoded inputs as follows. Let $\frac{1}{n} \sum_j W_{ij} = c_i$. We have:

$$\begin{aligned}
&b' + \sum_i \sum_j W'_{ij} X_{ij} \\
&= \left(b + \sum_i c_i \right) + \sum_i \sum_j \left(W_{ij} - c_i \right) X_{ij} \\
&= \left(b + \sum_i c_i \right) + \sum_i \left(\sum_j W_{ij} X_{ij} - c_i \sum_j X_{ij} \right) \\
&= \left(b + \sum_i c_i \right) + \sum_i \left(\sum_j W_{ij} X_{ij} - c_i \right) \text{ (Because } \sum_j X_{ij} = 1) \\
&= b + \sum_i \sum_j W_{ij} X_{ij} \\
&= y
\end{aligned}$$

The proof above is for the one-hot encoded case where the sum over all input channels at each position is 1 - however, note that a similar analysis could apply to any input constraint such that the sum over all channels at each position is a nonzero value.

References

[1] Pouya Kheradpour and Manolis Kellis. Systematic discovery and characterization of regulatory motifs in encode tf binding experiments. *Nucleic acids research*, 42(5):2976–2987, 2014.