Not Just A Black Box: Interpretable Deep Learning by Propagating Activation Differences

Avanti Shrikumar[1] (avanti@stanford.edu), **Peyton Greenside**[2] (pgreens@stanford.edu)
Anna Y. Shcherbina[2] (annashch@stanford.edu), **Anshul Kundaje**[1,3] (akundaje@stanford.edu)

1. Department of Computer Science, Stanford University, CA, USA
2. Biomedical Informatics, Stanford University, CA, USA
3. Department of Genetics, Stanford University, CA, USA

Abstract

A common criticism of neural networks is their lack of interpretability or "black box" nature. This is a major barrier to adoption in application domains such as biology, where interpretability is important. Here we present LIFTPAD (Learning Important Features Through Propagating Activation Differences), an intuitive and effective method for scoring the contributions of inputs to the output of a neural network. LIFTPAD compares the activation of each neuron to its 'reference activation', and assigns a contribution score to each of the neuron's inputs proportionally to how the inputs differ from their own 'reference activations'. We apply LIFTPAD to models trained on natural images as well as models trained on genomic sequences. We show that LIFTPAD correctly identifies important features in both cases and has significant advantages over gradient-based methods.

1. Introduction

As neural networks become increasingly popular, their reputation as a "black box" presents a barrier to adoption in fields where interpretability is paramount. Understanding the features that lead to a particular output builds trust with users and can lead to novel scientific discoveries. Simonyan et al. (2013) proposed using the gradients of a particular output with respect to the individual inputs to generate saliency maps, and showed the superiority of this approach to the deconvolutional nets approach of Zeiler et al. (2013). Guided backpropagation (Springenberg et al. 2014) is another variant which only considers gradients that have positive error signal. As shown in Figure 1, saliency maps can be substantially improved by simply multiplying the gradient with the input signal, which corresponds to a first-order Taylor approximation of how the output would change if the input were set to zero. For piecewise linear networks, the layerwise relevance propagation rules described in Samek et al. (2015) reduce to this approach (which doesn't suffer from the same numerical stability problems), assuming that bias terms are included in the relevance propagation (derivation not shown).

Gradient-based approaches are problematic because activation functions such as Rectified Linear Units (ReLUs) have a gradient of zero when they are not firing, and yet a ReLU that does not fire can still carry information - particularly if the associated bias is positive (which means that in the absence of any input, the ReLU does fire). Similarly, sigmoid or tanh activations are popular choices for the activation functions of gates in memory units of recurrent neural networks such as GRUs and LSTMs (Chung et al. 2014; Hochreiter and Schmidhuber 1997), but these activations have a near-zero gradient at high or low inputs even though such inputs can be very significant.

Here we present a general method for assigning feature importance that focuses on the difference between a neuron's activation and its 'reference' activation, where the reference activation is the activation that the neuron has when the network is provided a 'reference input'. The reference input is defined on a domain-specific basis according to what is appropriate for the task at hand. This circumvents the limitation of gradient-based approaches because, rather than relying on the local gradient at the point of activation, the activation is simply compared to its reference value. In the case of a ReLU that fires when provided the reference input (as is typically the case for ReLUs that have a positive bias), our method assigns a negative 'difference from reference' when the ReLU does not fire. Similarly, in the case of a sigmoidal unit which has an output of 0.5 when pro-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

vided the reference input, our method would assign a positive 'difference from reference' when the sigmoidal unit has an output near 1, even though the gradient at that output is negligible.

2. LIFTPAD Method

We denote the contribution of x to y as C_{xy}. Let the activation of a neuron n be denoted as A_n. Further, let the *reference* activation of neuron n be denoted A_n^0, and let the $A_n - A_n^0$ be denoted as δ_n. We define our contributions C_{xy} to satisfy the following properties.

2.1. Summation to δ

For any set of neurons S whose activations are minimally sufficient to compute the activation of y (that is, if we know the activations of S, we can compute the activation of y, and there is no set $S' \subset S$ such that S' is sufficient to compute the activation of y - in layman's terms, S is a full set of non-redundant inputs to y), the following property holds:

$$\sum_{s \in S} C_{sy} = \delta_y \tag{1}$$

That is, the sum over all the contributions of neurons in S to y equals the difference-from-reference of y.

2.2. Linear composition

Let O_x represent the output neurons of x. The following property holds:

$$C_{xy} = \sum_{o \in O_x} \frac{C_{xo}}{\delta_o} C_{oy} \tag{2}$$

In layman's terms, each neuron 'inherits' a contribution through its outputs in proportion to how much that neuron contributes to the difference-from-reference of the output.

2.3. Backpropagation Rules

We show that the contributions as defined above can be computed using the following rules (which can be implemented to run on a GPU). The computation is reminiscent of the chain rule used during gradient backpropagation, as equation 2 makes it possible to start with contribution scores of later layers and use them to find the contribution scores of preceding layers. To avoid issues of numerical stability when δ_n for a particular neuron is small, rather than computing the contribution scores explicitly, we instead compute *multipliers* m_{xy} that, when multiplied with the difference-from-reference, give the contribution:

$$m_{xy}\delta_x = C_{xy} \tag{3}$$

Let t represent the target neuron that we intend to compute contributions to, and let O_x represent the set of outputs of x. We show that:

$$m_{xt} = \sum_{y \in O_x} m_{xy} m_{yt} \tag{4}$$

The equation above follows from the linear composition property and the definition of the multipliers, as proved below:

$$\begin{aligned} C_{xt} &= \sum_{y \in O_x} \frac{C_{xy}}{\delta_y} C_{yt} \\ m_{xt}\delta_x &= \sum_{y \in O_x} \frac{C_{xy}}{\delta_y}(m_{yt}\delta_y) = \sum_{y \in O_x} C_{xy} m_{yt} \\ m_{xt} &= \sum_{y \in O_x} \frac{C_{xy}}{\delta_x} m_{yt} = \sum_{y \in O_x} m_{xy} m_{yt} \end{aligned} \tag{5}$$

In the equations below, I_y denotes the set of inputs of y.

2.3.1. Affine functions

Let

$$A_y = \left(\sum_{x \in I_y} w_{xy} A_x \right) + b \tag{6}$$

Then

$$m_{xy} = w_{xy} \tag{7}$$

Proof. We show that

$$\delta_y = \sum_{x \in I_y} m_{xy} \delta_x \tag{8}$$

Using the fact that $A_n = A_n^0 + \delta_n$, we have:

$$\begin{aligned} (A_y^0 + \delta_y) &= \left(\sum_{x \in I_y} w_{xy}(A_x^0 + \delta_x) \right) + b \\ &= \left(\sum_{x \in I_y} w_{xy} A_x^0 \right) + b + \sum_{x \in I_y} w_{xy}\delta_x \end{aligned} \tag{9}$$

We also note that the reference activation A_y^0 can be found as follows:

$$A_y^0 = \left(\sum_{x \in I_y} w_{xy} A_x^0 \right) + b \tag{10}$$

Thus, canceling out A_y^0 yields:

$$\delta_y = \sum_{x \in I_y} w_{xy}\delta_x = \sum_{x \in I_y} m_{xy}\delta_x \tag{11}$$

2.3.2. Max operation

We consider the case of max operation:

$$A_y = \max_{x \in I_y} A_x \tag{12}$$

Then we have:

$$m_{xy} = \mathbf{1}\{A_x = A_y\}\frac{\delta_y}{\delta_x} \tag{13}$$

Where $\mathbf{1}\{\}$ is the indicator function. If a symbolic computation package is used, then the gradient of y with respect to x can be used in place of $\mathbf{1}\{A_x = A_y\}$.

Proof.

$$\begin{aligned}\sum_{x \in y} m_{xy}\delta_x &= \left(\sum_{x \in y} \mathbf{1}\{A_x = A_y\}\frac{\delta_y}{\delta_x}\right)\delta_x \\ &= \sum_{x \in y} \mathbf{1}\{A_x = A_y\}\delta_y = \delta_y\end{aligned} \tag{14}$$

2.3.3. Other activations

The following choice for m_{xy}, which is the same for all inputs to y, satisfies summation-to-delta:

$$m_{xy} = \frac{\delta_y}{\sum_{x' \in I_y} \delta_{x'}} \tag{15}$$

This rule may be used for nonlinearities like ReLUs, PReLUs, sigmoid and tanh (where y has only one input), or for element-wise products. Situations where the denominator is near zero can be handled by applying L'hopital's rule, because by definition:

$$\delta_y \to 0 \text{ as} \sum_{x \in I_y} \delta_x \to 0 \tag{16}$$

2.4. A note on final activation layers

Activation functions such as a softmax or a sigmoid have a maximum δ of 1.0. Due to the *summation to δ* property, the contribution scores for individual features are lower when there are several redundant features present. As an example, consider $A_t = \sigma(A_y)$ (where $sigma$ is the sigmoid transformation) and $A_y = A_{x_1} + A_{x_2}$. Let the default activations of the inputs be $A^0_{x_1} = A^0_{x_2} = 0$. When $x_1 = 100$ and $x_2 = 0$, we have $C_{x_1t} = 0.5$. However, when both $x_1 = 100$ and $x_2 = 100$, we have $C_{x_1t} = C_{x_2t} = 0.25$. To avoid this attenuation of contribution in the presence of redundant inputs, we can use the contributions to y rather than t; in both cases, $C_{x_1y} = 100$.

2.5. A note on Softmax activation

Let $t_1, t_2 ... t_n$ represent the output of a softmax transformation on the nodes $y_1, y_2 ... y_n$, such that:

$$A_{t_i} = \frac{e^{A_{y_i}}}{\sum_{i'=1}^{n} e^{A_{y'_i}}} \tag{17}$$

Here, $A_{y_1} ... A_{y_n}$ are affine functions of their inputs. Let x represent a neuron that is an input to $A_{y_1} ... A_{y_n}$, and let w_{xy_i} represent the coefficient of A_x in A_{y_i}. Because $A_{y_1} ... A_{y_n}$ are followed by a softmax transformation, if w_{xy_i} is the same for all y_i (that is, x contributes equally to all y_i), then x effectively has zero contribution to A_{t_i}. This can be observed by substituting $A_{y_i} = w_{xy_i}A_x + r_{y_i}$ in the expression for A_{t_i} and canceling out $e^{w_{xy_i}A_x}$ (here, r_{y_i} is the sum of all the remaining terms in the affine expression for A_{y_i})

$$\begin{aligned}A_{t_i} &= \frac{e^{A_{y_i}}}{\sum_{i'=1}^{n} e^{A_{y'_i}}} = \frac{e^{w_{xy_i}A_x + r_{y_i}}}{\sum_{i'=1}^{n} e^{w_{xy_{i'}}A_x + r_{y_{i'}}}} \\ &= \frac{e^{w_{xy_i}A_x + r_{y_i}}}{\sum_{i'=1}^{n} e^{w_{xy_i}A_x + r_{y_{i'}}}} = \frac{e^{r_{y_i}}}{\sum_{i'=1}^{n} e^{r_{y_{i'}}}}\end{aligned} \tag{18}$$

As mentioned in the previous subsection, in order to avoid attenuation of signal for highly confident predictions, we should compute C_{xy_i} rather than C_{xt_i}. One way to ensure that C_{xy_i} is zero if w_{xy_i} is the same for all y_i is to mean-normalized the weights as follows:

$$\bar{w}_{xy_i} = w_{xy_i} - \frac{1}{n}\sum_{i'=1}^{n} w_{xy_{i'}} \tag{19}$$

This transformation will not affect the output of the softmax, but will ensure that the LIFTPAD scores are zero when a particular node contributes equally to all softmax classes.

2.6. Weight normalization for constrained inputs

Let y be a neuron with some subset of inputs S_y that are constrained such that $\sum_{x \in S_y} A_x = c$ (for example, one-hot encoded input satisfies the constraint $\sum_{x \in S_y} A_x = 1$, and a convolutional neuron operating on one-hot encoded rows has one constraint per column that it sees). Let the weights from x to y be denoted w_{xy} and let b_y be the bias of y. It is advisable to use normalized weights $\bar{w}_{xy} = w_{xy} - \mu$ and bias $\bar{b}_y = b_y + c\mu$, where μ is the mean over all w_{xy}. We note that this maintains the output of the neural net because, for any constant μ:

$$\begin{aligned} A_y &= \Big(\sum A_x(\bar{w}_{xy} - \mu)\Big) + (b_y + c\mu) \\ &= \Big(\sum A_x w_{xy}\Big) - \Big(\sum A_x \mu\Big) + (b_y + c\mu) \\ &= \Big(\sum A_x w_{xy}\Big) - c\mu + (b_y + c\mu) \\ &= \Big(\sum A_x w_{xy}\Big) + b_y \end{aligned} \tag{20}$$

The normalization is desirable because, for affine functions, the multipliers m_{xy} are equal to the weights w_{xy} and are thus sensitive to μ. To take the example of a convolutional neuron operating on one-hot encoded rows: by mean-normalizing w_{xy} for each column in the filter, one can ensure that the contributions C_{xy} from some columns are not systematically overestimated or underestimated relative to the contributions from other columns.

3. Results

3.1. Tiny ImageNet

A model with the VGG16 (Long et al., 2015) architecture was trained using the Keras framework (Chollet, 2015) on a scaled-down version of the Imagenet dataset, dubbed 'Tiny Imagenet'. The images were 64 × 64 in dimension and belonged to one of 200 output classes. Results shown in Figure 1; the reference input was an input of all zeros after preprocessing.

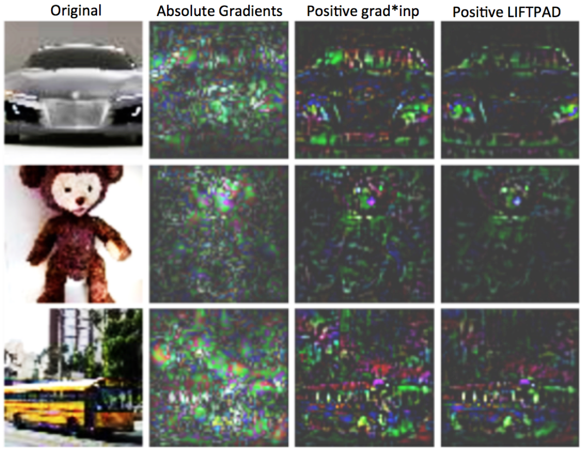

Figure 1. Comparison of methods to compute importance scores on Tiny ImageNet data. Left-to-right: original image, absolute value of the gradient, positive gradient*input (Taylor approximation), and positive LIFTPAD. LIFTPAD scores are less noisy than gradient*input.

3.2. Genomics

We apply LIFTPAD to models that classify genomic sequences. The positive class requires that two particular DNA patterns - 'GATA' and 'CAGATG' - appear anywhere in the length-200 sequence together. The negative class has just one of the two patterns appearing either once or twice. We simulate 20K sequences for the positive set and 40K sequences for the negative set split equally between the two patterns. Outside the core patterns (which were produced from a generative model) we randomly sample the remaining sequence with the four bases A, C, G and T. We trained a model using the Keras framework (Chollet, 2015) on one-hot encoded sequences with 20 convolutional filters of length 15 and stride 1 and a max pool layer of width and stride 50, followed by two fully connected layers of size 200. PReLU nonlinearities were used for the hidden layers. This model performs well with auROC of 0.907. The misclassified examples primarily occur when one of the patterns erroneously arises in the randomly sampled background.

We then run LIFTPAD to assign an importance score to each base in the correctly predicted sequences (the reference input for LIFTPAD was an input of all zeros) and compared the results to the gradient*input (Figure 2).

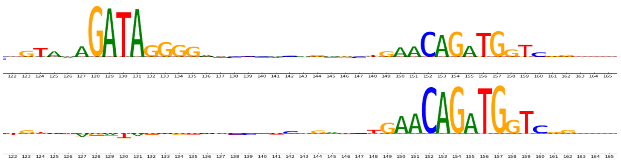

Figure 2. LIFTPAD scores (top) and gradient-based scores (bottom) using gradient*input are plotted for each position in the DNA sequence and colored by the DNA base (due to one-hot encoding, input is either 1 or 0; gradient*input is equivalent to taking the gradient for the letter that is actually present). LIFTPAD discovers both patterns and assigns them large importance scores. Gradient-based method miss the GATA pattern.

3.3. Author contributions

AS & PG conceived of LIFTPAD. AS implemented LIFTPAD in software. PG led application to genomics. AYS led application to Tiny Imagenet. AK provided guidance and feedback. AS, PG, AYS & AK prepared the manuscript.