

---

# LIFFTing the Top off the Black Box:

## Feature Selection for Neural Networks in Biology

---

### 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 DeepLIFT (Deep Linear Importance Feature Tracker), an intuitive and effective method for scoring the contributions of inputs to the output of a neural network. DeepLIFT compares the activation of each neuron to its ‘default activation’, and assigns a contribution score to each of the neuron’s inputs proportionally to how the inputs differ from their own ‘default activations’. We apply DeepLIFT to models trained on natural images as well as models trained on genomic sequences. We show that DeepLIFT 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. A common approach is to leverage the gradients of a particular output with respect to the individual inputs (Simonyan et al. - cite) - however, approaches are limited because commonly used activation functions such as Rectified Linear Units (ReLU) have a gradient of zero when they are not firing, 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 (cite), but these activations have a near-zero gradient at high or low inputs even though such inputs can be very significant.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

In this paper, we present a general method for assigning feature importance which focuses on the difference between a neuron’s activation and its ‘default’ activation, where the default activation is the activation that the neuron has when the network is provided a ‘default input’. The default input is defined on a domain-specific basis according to what is appropriate for the task at hand. This circumvents the aforementioned 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 default value. In the case of a ReLU which fires when provided the default input (as is typically the case for ReLUs that have a positive bias), our method assigns a negative ‘difference from default’ when the ReLU does not fire. Similarly, in the case of a sigmoidal unit which has an output of 0.5 when provided the default input, our method would assign a positive ‘difference from default’ when the sigmoidal unit has an output near 1, even though the gradient at that output is negligible.

### 2. DeepLIFT Method

We define the contributions of every neuron  $x$  to some downstream neuron  $y$ . 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 *default* activation of neuron  $n$  be denoted  $A_n^0$ , and let the  $A_n - A_n^0$  be denoted as  $\delta_n$ . We define our contributions  $C_{xy}$  to satisfy the following properties.

#### 2.1. Summation to $\delta$

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:

$$\sum_{s \in S} C_{sy} = \delta_y \quad (1)$$

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

## 2.2. Linear composition

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

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

In layman's terms, each neuron 'inherits' a contribution through its outputs in proportion to how much that neuron contributes to the difference-from-default 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 backpropagation, as equation 2 allows contribution scores to be computed efficiently via dynamic programming. To avoid issues of numerical stability when  $\delta_n$  for a particular neuron is small, rather than computing the contribution scores explicitly, we instead compute *multipliers*  $m_{xy}$  that satisfy the following condition:

$$m_{xy}\delta_x = C_{xy} \quad (3)$$

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:

$$m_{xt} = \sum_{y \in O_x} m_{xy}m_{yt} \quad (4)$$

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

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

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

### 2.3.1. LINEARITY

Let

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

Then

$$m_{xy} = w_{xy} \quad (7)$$

### Proof

We show that

$$\delta_y = \sum_{x \in I_y} m_{xy} \delta_x \quad (8)$$

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

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

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

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

Thus, canceling out  $A_y^0$  yields:

$$\begin{aligned} \delta_y &= \sum_{x \in I_y} w_{xy} \delta_x \\ \delta_y &= \sum_{x \in I_y} m_{xy} \delta_x \end{aligned} \quad (11)$$

### 2.3.2. POOLING

### 2.3.3. NONLINEARITY

## 2.4. A note on weight normalization for constrained inputs

## 2.5. A note on softmax activations