Axiomatic Attribution for Deep Networks

Mukund Sundararajan, Ankur Taly, Qiqi Yan

arXiv: https://arxiv.org/abs/1703.01365

The objective is to define a new method for attributing the prediction of a deep network to its input features. For example, such a method could tell us which pixels of the image were responsible for a certain label being picked, helping us understand the input-output behavior of the deep network, which gives us the ability to improve it or a give a rational for a given prediction or recommendation.

Challenges: A significant challenge in designing an attribution technique is that they are hard to evaluate empirically, since it is hard to tease apart errors that stem from the misbehavior of the model versus the misbehavior of the attribution method.

→ The paper identify two axioms that every attribution method must satisfy.

→ Show that most previous methods do not satisfy one of these two axioms

→ Use the axioms to identify a new method, called **integrated gradients.**

Formal Definition: suppose we have a function $F : R^n \rightarrow [0, 1]$ that represents a deep network, and an input $x = (x_1, ..., x_n) \in R^n$. An attribution of the prediction at input x relative to a baseline input x' s a vector $AF(x, x') = (a_1, ..., a_n) \in R^n$ where a_i is the contribution of x_i to the prediction $F(x)$

Two Fundamental Axioms

Axiom: Sensitivity(a)

An attribution method satisfies Sensitivity(a) if for every input and baseline that differ in one feature but have different predictions then the differing feature should be given a non-zero attribution.

Axiom: Implementation Invariance

Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. Attribution methods should satisfy Implementation Invariance, i.e., the attributions are always identical for two functionally equivalent networks.

Integrated Gradients

Suppose we have a function $F : R^n \rightarrow [0, 1]$ that represents a deep network. Specifically, let $x \in R^n$ be the input at hand, and $x' \in R^n$ be the baseline input. For image networks, the baseline could be the black image, while for text models it could be the zero embedding vector.

We consider the straight line path (in R^n) from the baseline x' to the input x, and compute the gradients at all points along the path. Integrated gradients are obtained by accumulating these gradients.

Formally, the integrated gradient along the i-th dimension for an input x and baseline x′ is defined as follows.

$$\text{IntegratedGrads } i(x) ::= (xi - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F\left(x' + \alpha \times \left(x - x'\right)\right)}{\partial x_i} d\alpha$$

Instead of choosing the straight line path, other paths could also be chosen:

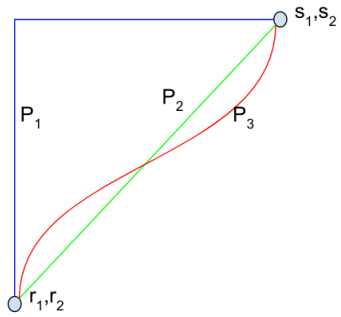

Figure 1. Three paths between an a baseline (r_1, r_2) and an input (s_1, s_2). Each path corresponds to a different attribution method. The path P_2 corresponds to the path used by integrated gradients.

The choice of the straight line path gives rise to symmetry-preserving attribution function

Applying Integrated Gradients

- Selecting a Benchmark: A key step in applying integrated gradients is to select a good baseline.We recommend that developers check that the baseline has a near-zero score.
  - For an object recognition network it is possible to create an adversarial example that has a zero score for a given input label. The attributions can then include undesirable artifacts of this adversarially constructed baseline.
  - In an object recognition network, a black image signifies the absence of objects. The black image isn't unique in this sense—an image consisting of noise has the same property. However, using black as a baseline may result in cleaner visualizations of "edge" features.
  - For text based networks, we have found that the all-zero input embedding vector is a good baseline. The action of training causes unimportant words tend to have small norms, and so, in the limit, unimportance corresponds to the all-zero baseline.
- Computing Integrated Gradients: The integral of integrated gradients can be efficiently approximated via a summation. We simply sum the gradients at points occurring at sufficiently small intervals along the straight line path from the baseline x′ to the input x.

  $$\text{IntegratedGrads } i^{approx}(x) ::= (xi - x_i') \times \Sigma_{k=1}^{m} \frac{\partial F\left(x' + \frac{k}{m} \times \left(x - x'\right)\right))}{\partial x_i} \times \frac{1}{m}$$

  Here m is the number of steps in the Riemman approximation of the integral. Notice that the approximation sim-ply involves computing the gradient in a for loop which should be straightforward and efficient in most deep learning frameworks. For instance, in TensorFlow, it amounts to calling tf.gradients in a loop over the set of inputs (i.e., $x' + \frac{k}{m} \times (x - x')$ for k= 1,...,m),

Examples

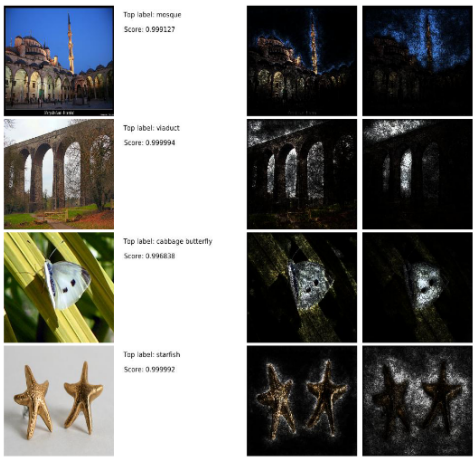

Figure 2. **Comparing integrated gradients with gradients at the image.** Left-to-right: original input image, label and softmax score for the highest scoring class, visualization of integrated gradients, visualization of gradients*image. Notice that the visualizations obtained from integrated gradients are better at reflecting distinctive features of the image.

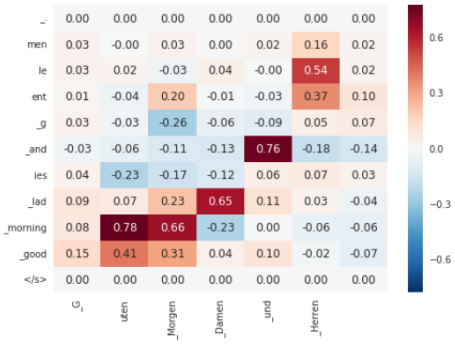

Figure 5. **Attributions from a language translation model.** Input in English: "good morning ladies and gentlemen". Output in German: "Guten Morgen Damen und Herren". Both input and output are tokenized into word pieces, where a word piece prefixed by underscore indicates that it should be the prefix of a word.

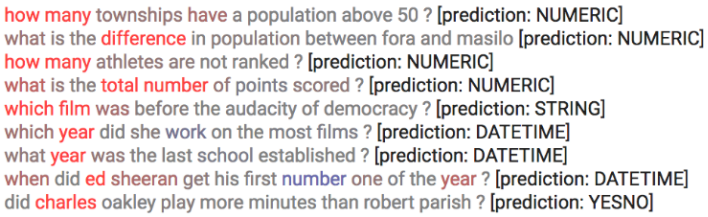

Figure 4. **Attributions from question classification model.** Term color indicates attribution strength—Red is positive, Blue is negative, and Gray is neutral (zero). The predicted class is specified in square brackets.