Gradient-Based Attribution Methods

Marco Ancona[1], Enea Ceolini[2], Cengiz Öztireli[1], and Markus Gross[1]

[1] Department of Computer Science, ETH Zurich, Switzerland
{anconam,cengizo,grossm}@inf.ethz.ch
[2] Institute of Neuroinformatics, University Zürich and ETH Zürich, Switzerland
eceoli@ini.uzh.ch

Abstract. The problem of explaining complex machine learning models, including Deep Neural Networks, has gained increasing attention over the last few years. While several methods have been proposed to explain network predictions, the definition itself of explanation is still debated. Moreover, only a few attempts to compare explanation methods from a theoretical perspective has been done. In this chapter, we discuss the theoretical properties of several attribution methods and show how they share the same idea of using the gradient information as a descriptive factor for the functioning of a model. Finally, we discuss the strengths and limitations of these methods and compare them with available alternatives.

Keywords: Attribution methods, Deep Neural Networks, Explainable Artificial Intelligence

1 Introduction

Machine Learning has demonstrated huge potential in solving a variety of problems, mostly thanks to numerous ground-breaking results with Deep Neural Networks (DNNs) [12, 14, 17, 29, 36]. As learning-based algorithms are deployed into everyday life products, explaining their predictions becomes of crucial importance, not only to ensure reliability and robustness, but also to make sure the prediction is fair and not discriminating. While machine learning models power an increasing number of applications, the black-box nature of DNNs has become a barrier to the adoption of these systems in those fields where interpretability is crucial. Whether machine learning is employed in medical prognosis, controlling a self-driving car or assessing the risk of committing new crimes, the decisions taken have a deep impact on the life of the people involved and precise guarantees need to be enforced. European regulations are rather strict in this sense, as they introduce the legal notion of a *right to explanation* [10], de facto banning the use of non-explainable machine learning models on some domains.

Interpretability in machine learning can be achieved in different ways. The first approach is to use models that are *inherently interpretable*, like linear models. Although this is the most immediate solution, these models often trade interpretability for limited predictive power. A second approach is to design models able to generate predictions and explanations, either in textual or visual

form, at the same time. Training this kind of models is very challenging as they require ground-truth explanations to train on. What is more, as the explanation generator is itself a trained model, the problem of evaluating fairness and accuracy is only being transferred from the operational model to the explanation model. Finally, one can build explanation methods on top of existing models. Potentially, this allows generating explanations for any existing black-box model without the need for retraining and without sacrificing the performances of the best models.

Another important dimension to consider is the *scope* of interpretability methods. *Global interpretability* is about understanding how the overall model makes decisions, which input patterns are captured and how these are transformed to produce the output [6]. Global interpretability is very useful to detect biases that might cause unfair or discriminating behavior of the model, but it is arguably very hard to achieve as the number of interacting parameters grows. For this reason, several works on interpretability of DNN models focus on explaining the reasons for a specific decision instead. This is commonly referred to as *local interpretability* [6, 15] as the model behavior is only explained for a single, specific instance. This type of explanation is useful to justify model predictions on a case by case basis, e.g. why a loan application was rejected.

In the remainder of this chapter, we focus on *attribution methods* for DNNs. As they are built on top of existing models, attribution methods belong to the last of the three explainability approaches discussed above. They aim at explaining decisions for existing neural network architectures that would be not explainable otherwise. Moreover, they operate in the scope of *local interpretability*, as they produce explanations specifically for a given model and input instance.

Several attribution methods have been developed specifically for neural networks [4, 18, 25, 27, 30, 32, 33, 37, 38]. However, the lack of a ground-truth explanation makes it very challenging to evaluate these methods quantitatively, while qualitative evaluations have been shown to be insufficient or biased [2, 8, 20]. With the proliferation of attribution methods, we believe a better understanding of their properties, assumptions and limitations becomes necessary.

We analyze some recently proposed attribution methods from a theoretical perspective and discuss connections between them. We show how the gradient with respect to the input features of a neural network is the fundamental piece of information used by several attribution methods to build explanations and we discuss the limitations of this approach.

The content of the chapter is mostly based on the results and considerations of a previous work [3], reorganized to gradually introduce the non-expert reader into the topic of attribution methods for DNNs. In Section 2, we formally define attribution methods, show how the gradient is used to generate attributions in a simple linear model and discuss some desirable properties of these methods. In Section 3, we move from linear models to DNNs, showing how the gradient information can be adapted to explain non-linear models. In Section 4, we discuss strengths, limitations and alternatives to gradient-based methods. Lastly, Section 5 presents some final considerations.

2 Attribution Methods

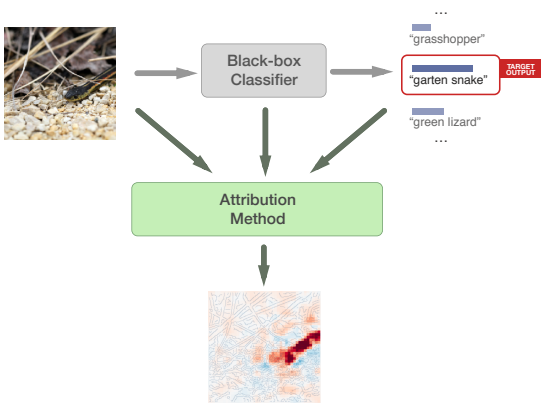


Fig. 1: General setting for attribution methods. An attribution map is generated for a specific specific input, model and target output.

Consider a model that takes an N-dimensional input $x = [x_1, ..., x_N] \in \mathbb{R}^N$ and produces a C-dimensional output $S(x) = [S_1(x), ..., S_C(x)] \in \mathbb{R}^C$. The model could be, for example, a DNN where C is the total number of output neurons. Depending on the application, the input features $x_1, ..., x_N$ can have different nature. For example, in image classification, x is usually a picture and each feature in x is a pixel in the picture. In the case of speech recognition, each feature might be the signal spectral power for a particular time and frequency bin. In the case of natural language processing, each feature is usually a multi-dimensional vector representation of each word in a sentence. Similarly, each output of the network can represent either a numerical predicted quantity (regression task) or the probability of a corresponding class (classification task).

Given their potentially large number, it is desirable to know which of the network input features have a greater impact on the predicted output, as well as the direction of their influence. Even though this is a simplistic notion of explanation, compared to the human notion which usually involves the formulation in textual form, it has been proved nevertheless useful to understand the reasons behind some network (mis)predictions [22].

Formally, attribution methods aim at producing explanations by assigning a scalar *attribution* value, sometimes also called "relevance" or "contribution", to each input feature of a network for a given input sample.

Given a single target output unit c, the goal of an attribution method is to determine the contribution $R^c = [R^c_1, ..., R^c_N] \in \mathbb{R}^N$ of each input feature x_i to the output $S_c(x)$. For a classification task, the target output can be chosen to be the one associated with the label of the correct class (e.g. to verify which input

features activated the class) or the one associated with a different class (e.g. to analyze the cause of a wrong classification).

Most often, attributions are visualized as *attribution maps*. These are heatmaps where, for each input feature, a red or blue color indicates the positive or negative influence of the feature to the target output, respectively (Figure 1). Therefore, attribution maps highlight features that are positively contributing to the target activation as well as features that have a suppressing effect on it (counter-evidence).

2.1 Example of attributions for a linear model

Before moving into non-linear models like DNNs, we show how attributions can be found in the linear case. This will also give us the opportunity to introduce the role of the gradient in generating attributions.

Linear models have been used for decades by statisticians, economists, computer and data scientists to tackle quantitative problems. Among such models, *Multiple Linear Regression* [13] can be used to model the dependency of a regression target y on a number of input features $x_1, ..., x_N$:

$$y = w_0 + w_1 x_1 + ... + w_N x_N + \epsilon, \tag{1}$$

where the w_i are the learned model weights and ϵ is the residual error. The model weights can be estimated by ordinary least squares (OLS). As the model optimization is not in the scope of this chapter, we refer the reader to [13] for further details. Under the assumptions of independence, linearity, normality and homoscedasticity, the weights of a linear regression model are easy to understand and provide an immediate tool for the model interpretation. Let us consider a minimal example, where a linear regression is used to estimate the future capital asset y_c, based on two investments x_1 and x_2. Let assume the assumptions above are met and the model parameters are estimated as follows:

$$\mathbb{E}[y_c|x_1, x_2] = 1.05x_1 + 1.50x_2, \tag{2}$$

We can derive immediately a *global* interpretation of this model. Every dollar invested in fund x_1 will produce a capital of 1.05$, while every dollar invested in x_2 will produce a capital of 1.50$, independently of the values x_1 and x_2 might assume in a concrete scenario. Notice that this explanation is purely based on the learned coefficient $w_1 = 1.05$ and $w_2 = 1.50$. These, sometimes called *partial regression coefficients*, are themselves candidate attribution values to explain the influence of the independent variables of the target variable:

$$R_1(x) = 1.05 \quad R_2(x) = 1.50 \tag{3}$$

Notice also that the coefficients are the partial derivatives of the target variable with respect to the independent variable, therefore this attribution is nothing but the model gradient:

$$R_i(x) = \frac{\partial y_c}{\partial x_i}(x) \tag{4}$$

On the other hand, this is not the only possible attribution scheme for such a model. Let us consider a concrete investment scenario where 100′000$ have been invested in x_1 and 10′000$ in x_2. The total capital asset, according to our trained model, will be 120′000$. One might be interested to know, in the particular case at hand, how x_1 and x_2 influenced the asset, i.e. how the two different investments contributed to the final capital. In this case, we are looking for a *local* explanation, as we are interested in explaining the response for a specific data point.

Again, thanks to the linearity of the model, it is easy to see that the 120′000$ can be explained as the sum of two contributions: 105′000$ from money invested in the first fund and 15′000$ from the money invested in the second one. Reasonable attribution values are, therefore:

$$R_1(x) = 105'000 \quad R_2(x) = 15'000 \tag{5}$$

In terms of relation with the gradient, we can formulate attributions as the gradient multiplied element-wise by the input:

$$R_i(x) = x_i \cdot \frac{\partial y_c}{\partial x_i}(x) \tag{6}$$

From this toy example, we have found two possible explanations in terms of attribution values. Notice that both are based on the gradient of the model function but significantly different. In the first case, the attribution for variable x_2 is significantly higher than for variable x_1, while in the second case the ranking is inverted. One might ask whether one of these is better than the other in explaining the model. As both attributions were derived with a clear objective, we argue that they are both reasonable explanations of the model. On the other hand, it is clear that they answer different questions: attributions in Equation 3 answer the question *"Where should one invest in order to generate more capital?*, while attributions in Equation 5 answer the questions *"How the initial investments contributed to the final capital?"*. Depending on the question that needs to be answered, the appropriate attribution methods should be chosen. As they answer different questions, it is also evident how the two attributions cannot be directly compared.

2.2 Salience versus Sensitivity methods

The two attribution methods described by Equations 4 and 6 differ in that the second one also involves the multiplication by the input itself. In the previous section, we discussed how they explain two very different aspects of the model. As we will discuss later, several (but not all) attribution methods for DNNs are computed by multiplying point-wise a quantity by the input being explained. The justification for multiplying by the input has been only partially discussed

in previous literature [28, 31, 33]. It has been noted that, when applied to image classification models, it contributes to making attribution maps sharper [28], although others noticed that it remains unclear how much of this can be attributed to the sharpness of the original image itself [31]. On the other hand, there is a more fundamental justification, which allows distinguishing attribution methods in two broad categories: *salience* and *sensitivity methods* [3].

A *sensitivity method* aims at describing how the output of the network changes when one or more input features are perturbed. In a linear model, this effect is constant for arbitrarily large perturbations. With non-linear models, sensitivity methods still represent the expected effect of a change in the input, but as first-order Taylor expansion of a non-linear function, they are only accurate for infinitesimal perturbations around the original input. While the model sensitivity to a feature perturbation has been regarded as a possible measure for the "importance" of the feature [30], the inherent objective of *sensitivity* methods should always be taken into account.

Conversely, a *salience method* describes the marginal effect of a feature to the output with respect to the same input where such feature has been removed. In other words, the produced values are meant to describe the contributions of the different input variables to the final target output. In this sense, for a *salience* method, it is often considered desirable that all attribution values sum up to the final target score, so that the attributions can be directly related to the output as additive contributions [4, 27, 33]. In our linear example, the multiplication between the input and the gradient provides a *salience* method that fulfills this property. We will see that the same method does not necessarily satisfy the property when extended to non-linear models. Notice also that *salience* methods must necessarily take the input into account when computing attributions.

2.3 Baseline for salience methods

Most *salience* methods require to define a baseline value. Since we are interested in the marginal effect of a feature, we are implicitly looking for how the output would change without that feature. As pointed out by others [33], humans also assign blame to a cause by comparing the outcomes of a process including or not such cause. In the particular case of a machine learning model, a feature with a non-zero attribution is expected to play some role in determining the output of the model.

Unfortunately, there is no proper way to *remove* one or more features from the input to a DNN, as most network architectures assume the number of input features is fixed. While we could re-train the network with a reduced set of input features, this would result in a different model whose internal mechanics are not necessarily related to the ones of the original model we want to explain.

In the related literature, this problem is often addressed by *simulating* the absence of a feature, instead of removing it in the strictest sense. First, a baseline input $\bar{x}$ is defined. The baseline should be chosen, depending on the domain, to best represent the absence of information (e.g. the black image). Then, attributions are computed with respect to the baseline, replacing each feature

with its corresponding baseline value anytime the absence of the feature needs to be tested.

An important consequence of this setup is that attributions are heavily affected by the choice of the baseline. In particular, explanations are a function of the difference between the input and the baseline, rather than the input alone. Unfortunately, how to choose an appropriate baseline for different domains is still an open research question. Moreover, the baseline must necessarily be chosen in the domain of the input space, which creates an evident ambiguity between a valid input that incidentally assumes the baseline value and the indicator for a missing feature.

For many attribution methods, the zero baseline is the canonical choice [27, 33, 37]. Sometimes, the zero baseline is also used implicitly by attribution methods that do not let the user define it [4, 22]. One possible justification for this particular choice relies on the observation that, for a model that implements a chain of operations of the form $z_j = f(\sum_i (w_{ji} \cdot z_i) + b_j)$, the all-zero input is somehow neutral to the output (i.e. $\forall c \in C : S_c(0) \approx 0$). If the model has no additive bias and all non-linear activations map zero to zero (e.g. ReLU and Tanh), the output is in fact zero when the network is fed a zero input. Empirically, the output is often near zero even when biases have different values, which makes the choice of zero for the baseline reasonable, although arbitrary.

Other choices are possible. Sometimes, the expected value of each feature over the training set is used as a baseline [16, 27]. In the image domain, previous works also suggested to set the baseline to a blurred version of the input images [7]. Finally, it is also possible to marginalize over the features to be removed in order to simulate their absence. For example, it has been shown how local coherence of natural images can be exploited to marginalize over image patches [38]. Unfortunately, this approach is significantly slower. What is more, it can only be applied to images or other domains where a prior about features correlation is known and can be exploited to make the problem computationally feasible.

2.4 Properties and Definitions

We now introduce some definitions and notable properties that will be used in the remainder of this chapter to characterize attribution methods.

Explanation continuity. We say an attribution method satisfies *explanation continuity* if, given a continuous prediction function to be explained $S_c(x)$, it produces continuous attributions $R^c(x)$. This is a desirable property for any attribution method [19]: if, for two nearly identical data points, the model response is nearly identical, then it is reasonable to expect that the corresponding explanations should also be nearly identical.

Implementation invariance. We say that two models m_1 and m_2 are *functionally equivalent* if, for any x provided as input to both models, they produce

the same output, despite possibly different implementations. More formally, if $\forall x : S_{m_1}(x) = S_{m_2}(x)$.

An attribution method is said to be *implementation invariant* [33] if it always produces identical attributions for functionally equivalent models m_1, m_2 provided with identical input :

$$\forall(m_1, m_2, x, c) : R^{c,m_1}(x) = R^{c,m_2}(x) \tag{7}$$

Sensitivity-n An attribution method satisfies *sensitivity-n* [3] when the sum of the attributions for any subset of features of cardinality n is equal to the variation of the output S_c caused removing the features in the subset. In this context, removing a feature means setting it to a baseline value, often chosen to be zero as discussed in Section 2.3. Mathematically, a method satisfies *sensitivity-n* when, for all subsets of features $x_S = [x_1, ...x_n] \subseteq x$, it holds:

$$\sum_{i=1}^{n} R_i^c(x) = S_c(x) - S_c(x \setminus x_S), \tag{8}$$

where the notation $x \setminus x_S$ indicates a data point x where all features in x_S have been replaced by a baseline value.

When $n = N$, with N being the total number of input features, we have $\sum_{i=0}^{N} R_i^c(x) = S_c(x) - S_c(\bar{x})$, where $\bar{x}$ is an input baseline representing an input from which all features have been removed. This property is known as *efficiency* in the context of cooperative game theory [23] and recognized as desirable for various attribution methods [4, 27, 33]. In related literature, the same property has been variously called *completeness* [33] or *summation to delta* [27].

Most often, the baseline is chosen to be neutral with respect to the model response ($S_c(\bar{x}) \approx 0$), in which case *sensitivity-N*, reduces to

$$\forall x, c : \sum_{i=1}^{N} R_i^c(x) = S_c(x) \tag{9}$$

meaning that the output of a model for a specific input x can be decomposed as sum of the individual contributions of the input features.

Clearly, this property only applies to *salience* methods. A notable observation is that no attribution method, as defined in Section 2, can satisfy *sensitivity-n* for *all* values of n, when applied to a non-linear model. The proof is provided in [3]. Intuitively, this is due to the fact that attribution maps have not enough degrees of freedom to capture non-linear interactions: given a non linear model, there must exists two features x_i and x_j such that $S(x) - S(x \setminus x_i, x_j) \neq 2 \cdot S(x) - S(x \setminus x_i) - S(x \setminus x_j)$. In this case, either *sensitivity-1* or *sensitivity-2* must be violated since attribution methods assign a single attribution value to both x_i and x_j.

3 Gradient-Based Attribution Methods for DNNs

In this section, we discuss how the idea of using the gradient information to generate attributions has been applied in several ways, implicitly or explicitly, to non-linear models, in particular DNNs. In Section 2.1 we showed how attributions for a linear regression model are generated as a function of the gradient with respect to the input features. In this section, we discuss the role of gradient while introducing several popular attribution methods for DNNs. We show how these methods differ in the way the backpropagation rules are modified in order to take into account the non-linearity of the network function. We also discuss how some methods can be revisited as gradient-based methods, despite their different original formulation. This will enable a direct comparison between them as part of a unified gradient-based framework.

3.1 From linear to non-linear models

In the case of a linear regression, we found two possible attribution methods:

$$R_i^c(x) = \frac{\partial S_c(x)}{\partial x_i} \qquad R_i^c(x) = x_i \cdot \frac{\partial S_c(x)}{\partial x_i} \tag{10}$$

We discussed how these methods can be both considered producing meaningful explanations for a linear model, although answering different questions. As a first attempt to produce explanations for a DNN, one might consider applying the same ideas. Indeed, this is what has been suggested in the related literature over the past decade.

Sensitivity analysis was one of the first methods to be adapted to the deep learning domain [30]. Attributions are constructed by taking the absolute value of the partial derivative of the target output S_c with respect to the inputs x_i:

$$R_i^c(x) = \left| \frac{\partial S_c(x)}{\partial x_i} \right| \tag{11}$$

Intuitively, the absolute value of the gradient indicates those input features (pixels, for image classification) that can be perturbed the least in order for the target output to change the most, discarding any information about the direction of this change. Nevertheless, Sensitivity analysis is usually rather noisy [18, 24, 31] and taking the absolute value prevents the detection of positive and negative evidence that might be present in the input.

Gradient * Input [28] was initially proposed as a technique to improve the sharpness of attribution maps generated by Sensitivity analysis. The attribution is computed by taking the (signed) partial derivatives of the output with respect to the input and multiplying them feature-wise by the input itself.

$$R_i^c(x) = \frac{\partial S_c(x)}{\partial x_i} \cdot x_i \tag{12}$$

The reader might recognize in Equation 12 an attribution method that we have previously discussed for linear models. While the multiplication with the

input can be theoretically justified if we are interested in *salience* instead of *sensitivity*, as we move from linear to non-linear models, computing *salience* is not as easy. Both Sensitivity analysis and Gradient * Input present clear shortcomings, as the partial derivative $S_c(x)/\ \partial x_i$ varies not only with x_i but also with the value of other input features.

Sensitivity analysis still provides a valid measure of the variation of the target output for a perturbation of the input variables but, in the case of a non-linear model, this value is only accurate for infinitesimally small perturbations around the original input. In fact, the resulting attributions can be seen as the first-order term of a Taylor decomposition of the function implemented by the network, computed at a point *infinitesimally close* to the actual input [18].

Similarly, as the gradient is not constant, Gradient * Input does not necessarily represent the correct marginal effect of a feature. Let us consider an additive model of the form:

$$y = w_0 + f_1(x_1) + ... + f_N(x_N) \tag{13}$$

where f_i are non-linear functions. Since the target y is computed as the sum of contributions of the input variables and there is no cross-interaction between them, we can expect a salience method to detect these contributions exactly and satisfy *sensitivity-N*. Concretely, let us consider the toy example $y = x_1 + \sqrt{x_2}$, evaluated at input $(4, 4)$. In this case, the contributions to the output ($y = 6$) would be naturally distributed as attributions $R_1 = 4$ and $R_2 = 2$. It is easy to verify that Gradient * Input fails to satisfy *sensitivity-N*, producing attributions $R_1 = 4$ and $R_2 = 1$, which do not sum up to the target value and are hardly justifiable. In order to overcome these limitations, gradient-based methods have been adapted to take into account the non-linear nature of DNNs.

3.2 Towards average gradients

Notice that in order to produce the expected attributions for the toy example illustrated above, we could simply replace the instant gradient, as in Gradient * Input, with the *average gradient* between the baseline (here $(0, 0)$) and the input value (here $(4, 4)$). This idea, in fact, is at the core of more recent attribution methods: Layer-wise Relevance Propagation (in its variant ϵ-LRP), DeepLIFT Rescale and Integrated Gradients. We now introduce these methods and discuss how they can all be examined under the same gradient-based framework. Notice that the following *post-hoc* considerations might not have been the ones that originally led to the formulation of these methods.

Layer-wise Relevance Propagation (ϵ-LRP) [4] is computed with a backward pass on the network. Let us consider a quantity $r_i^{(l)}$, called "relevance" of unit i of layer l. The algorithm starts at the output layer L, assigning the relevance of the target neuron c equal to the activation of the neuron itself, and the relevance of all other neurons to zero (Equation 14). Then it proceeds layer

by layer, redistributing the prediction score S_i until the input layer is reached. One recursive rule for the redistribution of a layer's relevance to the following layer is the ϵ-rule described in Equation 15, where we defined $z_{ij} = x_i^{(l)} w_{ij}^{(l,l+1)}$ to be the weighted activation of a neuron i onto neuron j in the next layer and b_j the additive bias of unit j. A small quantity ϵ is added to the denominator to avoid numerical instabilities. Once reached the input layer, the final attributions are defined as $R_i^c(x) = r_i^{(1)}$.

$$r_i^{(L)} = \begin{cases} S_i(x) & \text{if unit } i \text{ is the target unit of interest} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$r_i^{(l)} = \sum_j \frac{z_{ij}}{\sum_{i'} z_{i'j} + b_j + \epsilon \cdot sign(\sum_{i'} z_{i'j} + b_j)} r_j^{(l+1)} \tag{15}$$

LRP, together with the propagation rule described in Equation 15, is called ϵ-LRP, analyzed in the remainder of this chapter. We will briefly discuss a more recent variation of LRP in Section 4.3. Additionally, we assume a small and fixed ϵ, with the only purpose of avoiding divisions by zero.

We introduced the method according to the original formulation, where the role of the gradient might not appear immediately evident. On the other hand, we can reformulate ϵ-LRP to show how this method actually computes the average gradient in some cases and, based on this, discuss its advantages compared to Gradient * Input, as well as its limitations.

In a DNN where each layer performs a linear transformation $z_j = \sum_i w_{ji} x_i + b_j$ followed by a nonlinear mapping $x_j = f(z_j)$, a path connecting any two units consists of a sequence of such operations. The chain rule along a single path is therefore the product of the partial derivatives of all linear and nonlinear transformations along the path. For two units i and j in *subsequent* layers we have $\partial x_j / \partial x_i = w_{ji} \cdot f'(z_j)$, whereas for any two generic units i and c connected by a set of paths P_{ic} the partial derivative is the sum of the product of all weights w_p and all derivatives of the nonlinearities $f'(z)_p$ along each path $p \in P_{ic}$. We introduce a notation to indicate a modified chain rule, where the derivative of the nonlinearities $f'()$ is replaced by a generic function $g()$:

$$\frac{\partial^g x_c}{\partial x_i} = \sum_{p \in P_{ic}} \left(\prod w_p \prod g(z)_p \right) \tag{16}$$

When $g() = f'()$, Equation 16 is the definition of partial derivative of the output of unit c with respect to unit i, computed as the sum of contributions over all paths connecting the two units. Given that a zero weight can be used for non-existing or blocked paths, this is valid for any architecture that involves fully-connected, convolutional or recurrent layers without multiplicative units, as well as for pooling operations.

Notably, given this notation, ϵ-LRP can be reformulated as in the following proposition [3]:

Proposition 1 *ϵ-LRP is equivalent to the feature-wise product of the input and the modified partial derivative $\partial^g S_c(x)/\partial x_i$, with $g = g^{LRP} = f_i(z_i)/z_i$, i.e. the ratio between the output and the input at each nonlinearity.*

The reader can refer to Table 1 for the mathematical formulation, which makes the relation between ϵ-LRP and Gradient * Input more evident. In both cases, the input is multiplied with a quantity that depends on the network and its parameters. In the case of Gradient * Input, this quantity is simply the derivative of the output with respect to the input. In the case of ϵ-LRP, the quantity can be seen as a *modified* gradient, where the instant gradient of each nonlinearity in the chain rule is replaced by the ratio between the output and the input to the nonlinearity.

Furthermore, notice that $g^{LRP}(z) = (f(z) - 0)/(z - 0)$ which, in the case of Rectified Linear Unit (ReLU) or Tanh activations, is the *average gradient* of the nonlinearity in $[0, z]$. It is also easy to see that $\lim_{z\to 0} g^{LRP}(z) = f'(0)$, which explains why g, for these nonlinearities, can not assume arbitrarily large values as $z \to 0$, even without a stabilizer. On the contrary, if the discussed condition on the nonlinearity is not satisfied, for example with Sigmoid or Softplus activations, ϵ-LRP fails to produce meaningful attributions [3]. This is likely due to the fact $g^{LRP}(z)$ can become extremely large for small values of z, being its upper-bound only limited by the stabilizer. As a consequence, attribution values tend to concentrate on a few features. Notice also that the interpretation of g^{LRP} as average gradient of the nonlinearity does not hold in this case.

Method	Attribution $R_i^c(x)$
Sensitivity analysis	$\left\lvert \frac{\partial S_c(x)}{\partial x_i} \right\rvert$
Gradient * Input	$x_i \cdot \frac{\partial S_c(x)}{\partial x_i}$
ϵ-LRP	$x_i \cdot \frac{\partial^g S_c(x)}{\partial x_i}, \quad g = \frac{f(z)}{z}$
DeepLIFT (Rescale)	$(x_i - \bar{x_i}) \cdot \frac{\partial^g S_c(x)}{\partial x_i}, \;\; g = \frac{f(z) - f(\bar{z})}{z - \bar{z}}$
Integrated Gradients	$(x_i - \bar{x_i}) \cdot \int_{\alpha=0}^{1} \left. \frac{\partial S_c(\tilde{x})}{\partial(\tilde{x_i})} \right\rvert_{\tilde{x}=\bar{x}+\alpha(x-\bar{x})} d\alpha$

Table 1: Gradient-based formulation of several attribution methods.

DeepLIFT Rescale The use of average gradients to compute attributions has been later generalized by other methods. DeepLIFT [27] proceeds in a backward

fashion, similarly to LRP. Each unit i is assigned an attribution that represents the relative effect of the unit activated at the original network input x compared to the activation at some reference input $\bar{x}$ (Equation 17). Reference values $\bar{z}_{ij}$ for all hidden units are determined by running a forward pass through the network, using the baseline $\bar{x}$ as input, and recording the activation of each unit. The baseline is a user-defined parameter often chosen to be zero for the reasons discussed in Section 2.3. Equation 18 describes the relevance propagation rule.

$$r_i^{(L)} = \begin{cases} S_i(x) - S_i(\bar{x}) & \text{if unit } i \text{ is the target unit of interest} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

$$r_i^{(l)} = \sum_j \frac{z_{ij} - \bar{z}_{ij}}{\sum_{i'} z_{i'j} - \sum_{i'} \bar{z}_{i'j}} r_j^{(l+1)} \tag{18}$$

In Equation 18, $\bar{z}_{ij} = \bar{x}_i^{(l)} w_{ij}^{(l,l+1)}$ is weighted activation of a neuron i onto neuron j when the baseline $\bar{x}$ is fed into the network. The attributions at the input layer are defined as $R_i^c(x) = r_i^{(1)}$. The rule here described ("Rescale rule") is used in the original formulation of the method and it is the one we will analyze in the remainder of the chapter.

Similarly to ϵ-LRP, we can reformulate DeepLIFT (Rescale) to highlight the role of the gradient [3]:

Proposition 2 *DeepLIFT (Rescale) is equivalent to the feature-wise product of the $x - \bar{x}$ and the modified partial derivative $\partial^g S_c(x)/\partial x_i$, with $g = g^{DL} = (f_i(z_i) - f_i(\bar{z}_i))/(z_i - \bar{z}_i)$, i.e. the ratio between the difference in output and the difference in input at each nonlinearity, for a network provided with some input x and some baseline input $\bar{x}$ defined by the user.*

Again, by comparing the formulation of Gradient * Input and DeepLIFT (Table 1), it easy to see how the latter replaces the instant gradient of each nonlinearity in the interval that goes from the baseline to the actual input value. In this case, the formulation as average gradient holds also for those nonlinearities that do not cross the origin, such as Sigmoid and Softplus, the reason why DeepLIFT (Rescale) could be considered a generalization of ϵ-LRP that does not assume a zero baseline or a particular shape for the nonlinearity.

The formulation in Table 1 also highlights some further connections between Gradient * Input, ϵ-LRP and DeepLIFT. Motivated by the fact that attribution maps for different gradient-based methods look surprisingly similar on several tasks, some conditions of equivalence or approximation have been be derived [3]:

Proposition 3 *ϵ-LRP is equivalent to i) Gradient * Input if only ReLUs are used as nonlinearities; ii) DeepLIFT (computed with a zero baseline) if applied to a network with no additive biases and with nonlinearities f such that $f(0) = 0$ (e.g. ReLU or Tanh).*

The first part of Proposition 3 comes directly as a corollary of Proposition 1 by noticing that for ReLUs the gradient at the nonlinearity f' is equal to g^{LRP} for all inputs. This relation has been previously proven in the literature [11, 28]. Similarly, we notice that, in a network with no additive biases and nonlinearities that cross the origin, the propagation of the baseline produces a zero reference value for *all* hidden units (i.e. $\forall i : \bar{z_i} = f(\bar{z_i}) = 0$). Then $g^{LRP} = g^{DL}$, which proves the second part of the proposition.

Integrated Gradients We have seen how DeepLIFT and, in some cases, ϵ-LRP can be seen as computing a backward pass through the network where the gradient of the nonlinearities is replaced by their average gradient. There is one fundamental problem with this approach though: the chain rule does not hold for discrete gradients in general. As a consequence, the quantity computed by replacing each instant gradient by an average gradient at each nonlinearity does not necessarily result in the average gradient of the function as a whole. Moreover, two different implementations of the same function might lead to different results when the chain rule is applied. In this case, we say that the attribution method fails to satisfy *implementation invariance* [33].

Integrated Gradients [33] can be considered a generalization of DeepLIFT, designed to satisfy *implementation invariance*. It builds on the same idea of previous methods, as it can be interpreted as computing attributions multiplying the input variable element-wise with the average partial derivative, as the input varies from a baseline $\bar{x}$ to its final value x.

However, in this case, the original gradient is used for the nonlinearities, thus preserving the validity of the chain rule. Attributions are defined as follows.

$$R_i^c(x) = x_i \cdot \int_{\alpha=0}^{1} \frac{\partial S_c(\tilde{x})}{\partial(\tilde{x_i})}\Big|_{\tilde{x}=\bar{x}+\alpha(x-\bar{x})} d\alpha \tag{19}$$

Integrated Gradients presents several interesting properties. First, as the gradient of a function only depends on the function itself and not on its implementation, Integrated Gradients has the notable property of being implementation invariant, a property which is not satisfied by ϵ-LRP and DeepLIFT in general. Secondly, it can be immediately applied to any network architecture as it only depends on the function gradient, which is easily obtained thanks to the automatic differentiation of frameworks like Tensorflow [1] or PyTorch [21]. Finally, it always satisfies *sensitivity-N*, meaning that the sum of the produced attributions sum up to the network output minus the output when the network is evaluated at the baseline. Notice that DeepLIFT also satisfies *sensitivity-N* by design, but only if the computational graph does not include multiplicative interactions. This is illustrated in the following counter-example: take two variables x_1 and x_2 and a the function $h(x_1, x_2) = ReLU(x_1 - 1) \cdot ReLU(x_2)$. One can easily show that, by applying the methods as described by Table 1, DeepLIFT does not satisfy *sensitivity-N* while Integrated gradients does [3].

[3] DeepLIFT has been designed specifically for feed-forward neural networks and therefore assumes no multiplicative interactions. The gradient-based formulation

One of the limitations of Integrated Gradients is its relatively high computational cost, as evaluating the average gradient requires to numerically evaluate an integral. This means one needs to evaluate the model and its gradient several times, with slightly different inputs, which is computationally expensive for large models. While Integrated Gradients computes the average partial derivative of each feature as the input varies from a baseline to its final value, DeepLIFT approximates this quantity in a single step by replacing the gradient at each nonlinearity with its average gradient. Although the chain rule does not hold in general for average gradients, it has been shown empirically that DeepLIFT is most often a good approximation of Integrated Gradients in the case of feed-forward architectures [3].

4 Discussion

In the previous section, we have introduced five gradient-based methods for DNNs. While Sensitivity analysis, Gradient * Input and Integrated Gradients belong to this category by construction, we have discussed how ϵ-LRP and DeepLIFT (Rescale) can also fit in the definition of gradient-based methods, as they are computed by applying the chain rule for gradients once the instant gradient at each nonlinearity is replaced with a function that depends on the method. We have also suggested a theoretical justification for using the gradient information to produce explanations, starting from linear models and moving to DNNs. In this section, we highlight the advantages and limitations of gradient-based methods and discuss some possible alternatives.

4.1 Advantages of gradient-based methods

The strength of gradient-based methods is twofold. First, they are fast. Sensitivity analysis, Gradient * Input and ϵ-LRP only require a single forward and backward pass through the network to produce the attribution map. DeepLIFT requires one more forward pass to set the baselines for all nonlinearities, an operation that is computationally negligible in most of the use-cases. Integrated Gradients is slower than the others as it requires 50-200 backward passes for the numerical evaluation of the integral. Nevertheless, compared to other, not gradient-based methods, it can be still considered very efficient. What is more, the number of network evaluations does not depend on the number of input features to the network, which allows these methods to scale more easily. Notice also that frameworks like Tensorflow and PyTorch provide optimized algorithms to compute the gradients, often evaluated on the GPU to reach the best performance.

The second advantage of gradient-based methods is the ease of implementation. As pointed out by others [33], a desirable property for attribution methods is their immediate applicability to existing models. When the pure gradient is used,

generalizes the method to other architectures but does not guarantee meaningful results outside the scope DeepLIFT was designed for.

like in the case of Gradient * Input or Integrated Gradients, attributions can be computed for any network architecture where a gradient is defined, with very few lines of code. Our gradient-based formulation makes this possible for ϵ-LRP and DeepLIFT (Rescale) as well. Since all modern frameworks for Deep Learning implement backpropagation for efficient computation of the chain rule, it is possible to implement all methods above by overriding the gradient of all nonlinearities in the computational graph, with no need to implement custom layers or operations. Listings 1.1 and 1.2 show an example of how to achieve this on Tensorflow, respectively for ϵ-LRP and DeepLIFT (Rescale).

```
@ops.RegisterGradient("GradLRP")
def _GradLRP(op, grad):
    op_out = op.outputs[0]
    op_in = op.inputs[0]
    return grad * op_out / (op_in + eps)
```

Listing 1.1: Example of gradient override for a Tensorflow operation. After registering this function as the gradient for nonlinear activation functions, a call to `tf.gradients()` and the multiplication with the input will produce the ϵ-LRP attributions.

```
@ops.RegisterGradient("GradDeepLIFT")
def _GradDeepLIFT(op, grad):
    op_out = op.outputs[0]
    op_in = op.inputs[0]
    delta_out = op_out - ref_output
    delta_in = op_in - ref_input
    if tf.abs(delta_in) < eps:
        return grad
    else:
        return grad * delta_out / delta_in
```

Listing 1.2: Example of gradient override for a Tensorflow operation (DeepLIFT (Rescale)). Compared to ϵ-LRP, this requires to compute a reference input and output for each hidden unit, with a second forward pass.

4.2 Limitations

Unfortunately, gradient-based methods are also strongly affected by noisy gradients, as depicted in Figure 2, which shows the attributions generated for a Convolutional Neural Network (CNN) performing image classification. While most attribution mass is assigned to the area of the picture with the main subject, which seems reasonable, the attribution value assigned to individual pixels is affected by high-frequency variations, with neighboring pixels often being assigned very different attributions, possibly of the opposite sign. This phenomenon is likely to be caused by the violation of *explanation continuity* on gradient-based methods.

To illustrate the problem, let us consider a simple continuous function $y = max(x_1, x_2)$ and compute the attributions for the two input variables (x_1, x_2) as the input varies from $(2, 2 + \epsilon)$ to $(2, 2 - \epsilon)$, assuming the default baseline $(0, 0)$. Notice that this kind of function is very common in CNNs, which often include several *max-pooling* layers. Notice also that the function is continuous as the output varies smoothly from $2 + \epsilon$ to 2. On the other hand, since the gradient for the *max* function acts as a hard switch, all gradient-based methods discussed above [4] present a discontinuity in $(2, 2)$ where the attribution mass suddenly moves from x_2 entirely to x_1, as shown in Table 2. Intuitively, assigning all the attribution to the input feature with the highest value ignores the fact that, without that feature, the output would still be significantly high thanks to the influence of the other. While there is no doubt that the two input variables have some interaction to generate the result, generally it is not straight-forward to find a fair mechanism for credit assignment. The problem has been extensively studied in the context of cooperative game theory, where alternative mechanism, often proven to be closer to the human intuition, have been proposed [26]. Unfortunately, these methods are also computationally expensive and only applicable to DNNs with some significant approximations [16], a reason why gradient-based methods are still very popular.

Attributions for $min(x_1, x_2)$

Input	$(x_1, x_2) = (2, 2 + \epsilon)$	$(x_1, x_2) = (2, 2 - \epsilon)$
Attributions	$(R_1, R_2) = (0, 2 + \epsilon)$	$(R_1, R_2) = (2, 0)$

Table 2: Example of attributions generated for a continuous function by gradient-based methods (Gradient * Input, Integrated Gradients and most implementations of ϵ-LRP and DeepLIFT (Rescale).

In general, continuity in the produced explanations is a desirable property for any explanation method [19]. When we move away from a single operation and consider more complex models such as deep CNNs, the violation of *explanation continuity* exacerbates the problem of shattered gradients [5]: a high number of piece-wise linear regions in the learned function causes the gradient to become highly discontinuous and resemble white noise, which turns into high-frequency variations in the attributions produced by gradient-based methods. This makes

[4] In fact, ϵ-LRP and DeepLIFT (Rescale) are not implementation invariant so the result might change depending on the actual function implementation adopted in the network. For example, it can be implemented as a primitive operation (*max-pooling*) or, for positive numbers, it can be implicitly implemented by a two-layer network with three hidden units: $y = 0.5 * (ReLU(x_1 - x_2) + ReLU(x_2 - x_1) + ReLU(x_1 + x_2)$. In both cases, our reference implementation [3] produces the same attributions for all gradient-based methods, including ϵ-LRP and DeepLIFT (Rescale).

these explanations particularly sensitive to small variations in the input and not necessarily representative of the overall classification process.

4.3 Beyond gradient-based methods

Gradient-based methods are certainly not the only ones proposed in the literature. In particular, the gradient-based methods discussed before can be framed within a broader category of *backpropagation methods*, where the influence of the input features on the output is estimated layer by layer, trying to reverse the flow of information of the forward pass.

Instead of considering the model as a black-box that cannot be inspected, backpropagation methods take advantage of lower level access to the model's computational components and of the knowledge of how information flows in a DNN. Most often, backpropagation methods compute attribution maps with a few forward and backward passes through the network (Fig. 3a). In the forward pass, normal inference is performed on the input that needs to be explained. Then, a backward pass is performed, starting from a target (output) unit and propagating its activation through the network, layer by layer, until the input layer is reached, where the attribution map is formed.

Backpropagation methods differ in the way information flows in the backward pass. While this might consist in computing the chain rule for gradients (possibly replacing the gradient of the nonlinearities with other quantities), this is certainly not the only possible way to propagate relevance information from upper layers to the input. Most notably, some recent variants of LRP and DeepLIFT are backpropagation methods that cannot be easily fit in the definition of gradient-based methods. For example, $\alpha\beta$-LRP [4, 19] employs a backpropagation rule where the positive and negative information paths are weighted according to two different parameters, chosen by the user. This adaptation enables *explanation continuity* [19] but it also produces attributions that diverge from other gradient-based methods, making it hard to apply the theoretical framework we discussed so far. Similarly, a second variant of DeepLIFT known as "RevealCancel" [27] inherits some considerations from cooperative game theory to fix the problem of *explanation continuity* as discussed in the previous section. Both of these methods produce interesting results but more research will be necessary to derive strong theoretical foundations for these propagation rules, whose heuristics might have limitations not fully understood.

Other notable attribution methods are the so-called *perturbation methods*. In this case, attributions for each input feature (or set of features) are computed by directly removing, masking or altering them, and running a second forward pass on the new input, measuring the effect that this operation has on the output (Fig. 3b).

Perturbation methods have the advantages of a straightforward interpretation, as they are a direct measure of the marginal effect of some input features to the output. They are also model-agnostic, meaning that they can be applied to any black-box model that provides an evaluation function, without any need to access the internal operations. These methods have been first applied to CNNs in the

domain of image classification, visualizing the probability of the correct class as a function of the position of a grey patch occluding part of the image, a method known as Box Occlusion [37]. Other methods are Prediction Difference Analysis [38], Meaningful Perturbation [7] and LIME [22].

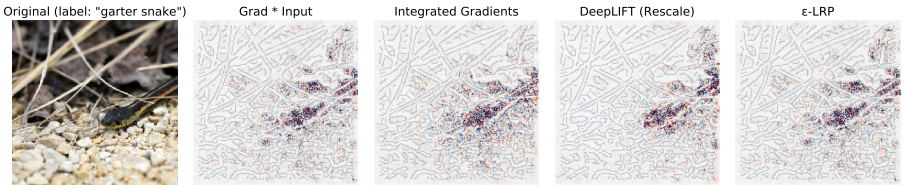
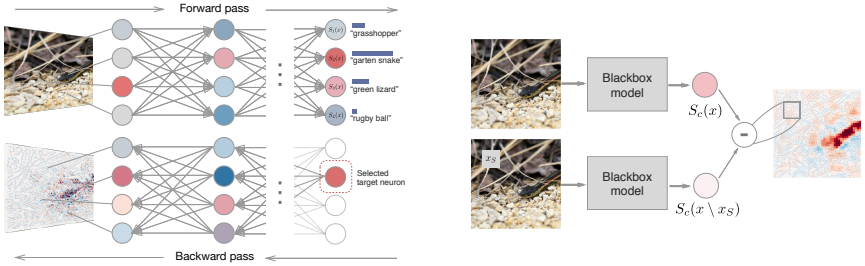


Fig. 2: Attribution generated by applying several attribution methods to an Inception V3 network for natural image classification [34]. Notice how all gradient-based methods produce attributions affected by significant local variance.



(a) Backpropagation methods (b) Perturbation methods

Fig. 3: Categorization of attribution methods

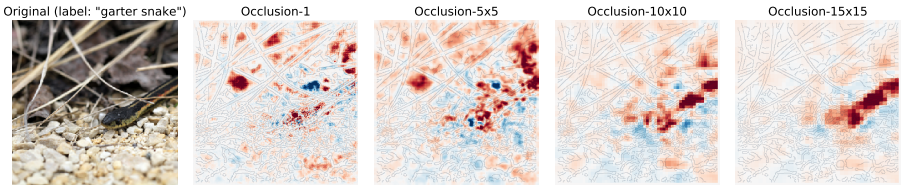


Fig. 4: Attributions generated by occluding portions of the input image with squared grey patches of different sizes. Notice how the size of the patches influence the result, with focus on the main subject only when using bigger patches.

A major limitation of perturbation methods is that the number of features that are perturbed altogether at each iteration, as well as the chosen perturbation technique, significantly affect the resulting explanations. Figure 4 highlights this problem in the case of Box Occlusion, where different explanations are generated for different sizes of the occluding box. While an ideal perturbation method would test all possible subsets of input features, this is not computationally feasible with most real-life datasets, as the cardinality of the power-set grows exponentially with the size of the input. In practice, perturbation methods differ by their underlying perturbation technique but the lack of a theoretically-grounded way of choosing it raises questions about the reliability of the resulting explanations.

Moreover, notice that perturbation methods tend to be significantly slower than gradient-based methods (up to hours for a single image [38]). As the number of required network evaluations grows with the number of input features, these are often aggregated to obtain attributions within a reasonable time. Which and how features are aggregated is also an hyper-parameter that strongly affects the resulting explanations.

5 Conclusions

Understanding and explaining complex machine learning models have become crucially important. While several attribution methods have been proposed in the last decade, we see no silver bullet among them. As discussed in the case of a linear model, the notion itself of "explanation" is not well defined, and therefore attribution methods sometimes answer different questions.

Moreover, the empirical evaluation of attribution methods is still hard to accomplish, as no fully reliable quantitative metrics are available. Therefore, when an explanation highlights some unexpected behavior of the model, it might be difficult to discern whether the problem is in fact due to the model itself or rather to the method used to explain the model [33].

Generally, the problem of validating attribution methods is accentuated by the lack of a solid theoretical foundation for the propagation rules of many backpropagation methods, as well as the little theoretical understanding of how different perturbation techniques affect the result on perturbation methods. Too often, attribution methods are only evaluated qualitatively, for example by comparing the attribution maps generated by different methods for the same model. We believe this is an extremely dangerous approach because humans might judge more favorably methods that produce explanations closer to their own expectations. Take, for example, a DNN trained for image classification: several works have shown that these networks are easily fooled by carefully-crafted adversarial samples, altered images almost indistinguishable from the originals by the human eye, but able to trick the network into a totally wrong classification [9, 35]. Adversarial samples suggest that DNNs are much more sensitive to small variations in the intensity of image pixels than the human brain is. So, if the mechanisms underlying the image recognition task in a DNN and in the human brain are different, we would expect the corresponding explanations to reflect

this difference. Instead, qualitative evaluations tend to be biased towards what we consider a "good" explanation as humans.

In order to develop better quantitative tools for the evaluation of attribution methods, we first need to define the goal that an ideal attribution method should achieve, as different methods might be suitable for different tasks. Then, as for the properties of *explanation continuity*, *implementation invariance* and *sensitivity-N*, we can proceed defining those axioms or properties we consider necessary for any good explanation method to fulfill. We believe this is a promising path to follow for future research on this topic.

To conclude, in this chapter we have analyzed five attribution methods and the role of the gradient in defining them. Starting from a linear model, we have shown how the gradient information can be used to generate explanations, explicitly or implicitly. We have also discussed some theoretical properties of these methods and shown that, despite their apparently different formulation, they are strongly related. By reformulating ϵ-LRP and DeepLIFT (Rescale), we have shown how these can be conveniently implemented with modern machine learning frameworks, similarly to other gradient methods. Finally, we have discussed some limitations of gradient-based methods, in the hope of encouraging further exploration of new techniques towards achieving explainable DNN models.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/

[2] Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., Kim, B.: Sanity checks for saliency maps. In: Advances in Neural Information Processing Systems. pp. 9524–9535 (2018)

[3] Ancona, M., Ceolini, E., Oztireli, C., Gross, M.: Towards better understanding of gradient-based attribution methods for deep neural networks. In: 6th International Conference on Learning Representations (ICLR) (2018)

[4] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015)

[5] Balduzzi, D., Frean, M., Leary, L., Lewis, J.P., Ma, K.W.D., McWilliams, B.: The shattered gradients problem: If resnets are the answer, then what is the question? In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. pp. 342–350. ICML'17, JMLR.org (2017)

[6] Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv (2017), https://arxiv.org/abs/1702.08608

[7] Fong, R.C., Vedaldi, A.: Interpretable explanations of black boxes by meaningful perturbation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3429–3437 (2017)

[8] Ghorbani, A., Abid, A., Zou, J.: Interpretation of neural networks is fragile. AAAI 2019 (2019)

[9] Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. International Conference on Learning Representations (ICLR) (2015)

[10] Goodman, B., Flaxman, S.: European union regulations on algorithmic decision-making and a "right to explanation". ICML Workshop on Human Interpretability in Machine Learning (WHI) (2016)

[11] Kindermans, P., Schütt, K., Müller, K., Dähne, S.: Investigating the influence of noise and distractors on the interpretation of neural networks. NIPS Workshop on Interpretable Machine Learning in Complex Systems (2016)

[12] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proc. of NIPS. pp. 1097–1105 (2012)

[13] Kutner, M.H., Nachtsheim, C., Neter, J.: Applied linear regression models. McGraw-Hill/Irwin (2004)

[14] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
[15] Lipton, Z.C.: The mythos of model interpretability. ICML Workshop on Human Interpretability of Machine Learning (2016)
[16] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Proceedings of Advances in Neural Information Processing Systems (NIPS). pp. 4765–4774 (2017)
[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
[18] Montavon, G., Lapuschkin, S., Binder, A., Samek, W., Müller, K.R.: Explaining nonlinear classification decisions with deep taylor decomposition. Pattern Recognition **65**, 211–222 (2017)
[19] Montavon, G., Samek, W., Müller, K.R.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73**(Supplement C), 1 – 15 (2018)
[20] Nie, W., Zhang, Y., Patel, A.: A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. ICML 2018 (2018)
[21] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS Autodiff Workshop (2017)
[22] Ribeiro, M.T., Singh, S., Guestrin, C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1135–1144. KDD '16, ACM, New York, NY, USA (2016)
[23] Roth, A.E.: The Shapley value: essays in honor of Lloyd S. Shapley. Cambridge University Press (1988)
[24] Samek, W., Binder, A., Montavon, G., Lapuschkin, S., Müller, K.R.: Evaluating the visualization of what a deep neural network has learned. IEEE Transactions on Neural Networks and Learning Systems (2016)
[25] Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 618–626 (2017)
[26] Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games **2**(28), 307–317 (1953)
[27] Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3145–3153. PMLR, International Convention Centre, Sydney, Australia (06–11 Aug 2017)
[28] Shrikumar, A., Greenside, P., Shcherbina, A., Kundaje, A.: Not just a black box: Learning important features through propagating activation differences. arXiv preprint arXiv:1605.01713 (2016)

[29] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)

[30] Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. ICLR Workshop (2014)

[31] Smilkov, D., Thorat, N., Kim, B., Viégas, F., Wattenberg, M.: SmoothGrad: removing noise by adding noise. ICML Workshop on Visualization for Deep Learning (2017)

[32] Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. ICLR 2015 Workshop (2015)

[33] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3319–3328. PMLR, International Convention Centre, Sydney, Australia (06–11 Aug 2017)

[34] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2818–2826 (2016)

[35] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (ICLR) (2014)

[36] Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. Transactions of the Association for Computational Linguistics **5**, 339–351 (2017)

[37] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision. pp. 818–833. Springer (2014)

[38] Zintgraf, L.M., Cohen, T.S., Adel, T., Welling, M.: Visualizing deep neural network decisions: Prediction difference analysis. In: International Conference on Learning Representations (2017)