Looking into Black Boxes: Techniques for Interpreting Neural Networks

Xisen Jin
University of Southern California
xisenjin@usc.edu

Abstract

Neural networks achieve impressive performance in various tasks, but they are usually treated black boxes. In this lecture, we discuss techniques for interpreting specific predictions or global behaviors of neural networks. We also discuss their application and evaluation in natural language processing.

1 Introduction

To start with, let's first consider what interpretability means and why we need it.

What is interpretability? Actually, there is hardly a mathematical definition for interpretability. A generally accepted definition is that interpretability is the degree to which a human can understand the cause of a decision (Molnar, 2019).

Why is interpretability useful? Interpretability is important in different ways. Some reasons are:

- In real word scenarios, user may expect an explanation for model decisions, e.g. item recommendation or online toxic comment detection.
- In high-risk scenarios, interpretation of models helps humans to detect biased behaviors of models.
- Interpretability of models enables knowledge extraction from models. For example, we can analyze data by looking at the coefficients regarding each feature of a linear regression model, because the model is inherently interpretable.

For better interpretability, the first choice is to build a *self-interpretable model*, such as linear-models or decision trees. Some neural network architectures like attention mechanism also bring some limited interpretability to models. An issue of attention mechanism is that the explanations are agnostic to classes and it becomes difficult to interpret predictions when there are stacked layers of attentions. There are also concerns about whether interpretable modules behaves as humans expect (Jain & Wallace, 2019).

The next choice is to employ post-hoc interpretation algorithms, which interpret a model with the model fixed. The algorithms either interpret a specific prediction, or interpret global behaviors of models. This lecture will cover several algorithms from each category. Table 1 shows some representative algorithm.

2 Methods

We first discuss techniques for interpreting a single prediction. These algorithms are usually referred to as *local explanation* algorithms. Let $\mathbf{x}$ be the input features for models, $f(\cdot)$ be the trained neural network, $y = f(\mathbf{x})$ be the prediction of the neural networks before the normalization layer (softmax,

CSCI 699 Lecture notes. October 29, 2019.

Reference	Method	TL;DR
Ribeiro et al. (2016)	LIME	Fit linear models around a data point
Li et al. (2016)	Occlusion	Mask an input and observe prediction differences
Simonyan et al. (2013)	Gradient	Plot gradients regarding outputs as saliency maps
Shrikumar et al. (2017)	DeepLIFT	Backprop activation difference as saliency maps
Sundararajan et al. (2017)	Integrated Gradients	Integrated gradients from reference as saliency maps
Lundberg & Lee (2017)	SHAP	Attribution inspired by Shapley Values
Chen et al. (2019)	L/C-Shapley	Shapley value approximation for structured data
Kim et al. (2017)	TCAV	Explanation using high-level concepts

Table 1: Overview of an incomplete list of interpretation techniques

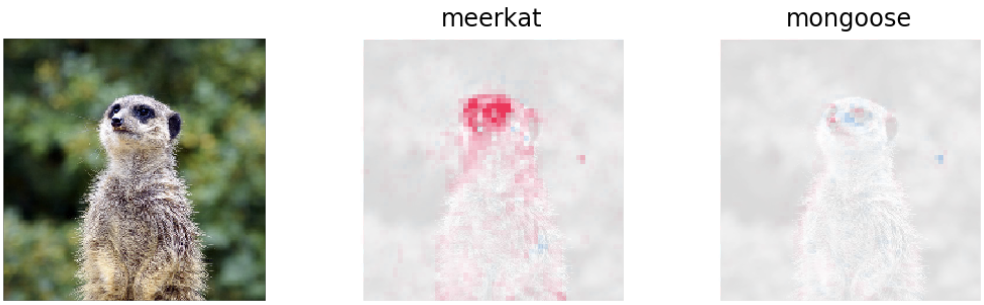


Figure 1: An example of local explanations on an input example for two classes. Red pixels increase the probability of the class, while blue pixels reduce the probability of the class. Image from `https://github.com/slundberg/shap`

sigmoid, etc.). To simplify the discussion, suppose the ground truth label for y is binary, such as sentiment analysis tasks. To interpret a specific prediction, an explanation algorithm attributes an importance score $\phi(x_i, \mathbf{x})$ for each input feature x_i (e.g. word, pixel) in $\mathbf{x}$.

2.1 Input Occlusion

A simple yet effective local explanation algorithm is input occlusion based approaches. Most prior works study input occlusion anecdotally and implement it by replacing the input features with padding values and observe prediction differences. Zintgraf et al. (2017) provide a mathematical formulation of input occlusion algorithms. It measures how the model prediction changes when a feature x_i is not "observed", that is, marginalized out.

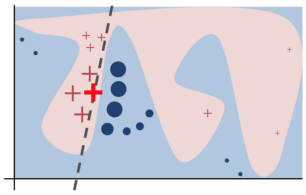
$$\phi(x_i, \mathbf{x}) = f(\mathbf{x}) - \mathbb{E}_{p(x_i|\mathbf{x}_{-i})}[f(\mathbf{x}_{-i}; x_i)] \tag{1}$$

The conditional expectation here is hard to model and is usually approximated. Zintgraf et al. (2017) experiment with (1) single-point estimation of unconditioned expectation by padding x_i with a reference value (2) padding x_i with the average pixel value around it. Chang et al. (2019) train a conditional GAN to fill in x_i.

A potential risk of padding based input occlusion algorithms is that it introduces artifacts, which means the input is significantly out of the normal data distribution. In NLP tasks, it breaks the grammar structure of a sentence. The model may perform unexpectedly on these data.

2.2 Local Surrogate (LIME)

Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016) interpret a prediction by fitting a local linear model around the data point. The algorithm first sample some data points around the input $\mathbf{x}$ by random perturbation. Then the models fits a linear model on sampled data points. The samples closer to the input $\mathbf{x}$ has a larger weight. For NLP tasks, there is a variant of LIME called Local Interpretable Model-agnostic Substring-based Explanations (LIMESSE) (Poerner et al., 2018). It first samples a substring length l, and sample a starting point x_s. LIMESSE use these sampled substrings to fit a linear model. See Figure 2 for illustrations.



(a)

	For	Christmas	Song	visit	my	channel!	;)	prob	weight
2	1	0	1	1	0	0	1	0.17	0.57
3	0	1	1	1	1	0	1	0.17	0.71
4	1	0	0	1	1	1	1	0.99	0.71
5	1	0	1	1	1	1	1	0.99	0.86
6	0	1	1	1	0	0	1	0.17	0.57

(b)

Figure 2: (a) Illustration of LIME. LIME fits a linear model around a data point. (b) The set of perturbed data points in spam classification task and the associated weights of examples. Recall that examples that are closer to the original input have a larger weight. By fitting a linear bag-of-words model on these data, we see the word "channel!" have a weight of 6.2, and other words have a weight of 0.

A significant difference between input occlusion and LIME is that LIME is an *additive attribution* method, i.e., the attribution for each input feature add up to the model prediction.

2.3 Shapeley Addtive Explanations (SHAP)

Intuitively, we seek explanation algorithms that attribute a "fair" importance scores for input features. To achieve this, we can first mathematically define some desirable properties that explanation algorithms should satisfy. Interestingly, sometimes we can find a *unique* solution that satisfies given properties. Shapeley Addtive Explanation (SHAP) is such an example.

SHAP formulate additive attribution algorithms in the perspective of cooperative game theory. Each feature acts as a *player*, and all the features form a *grand coalition* G. The neural networks f act as a *worth function*, which can be understood as the profits that a coalition of players make, that can be evaluate on any subset of players S. The features in $\mathbf{x}$ that are not included in the subset S are marginalized out. We note the feature after marginalization as $\mathbf{x}_S$. The *marginal contribution* of a player i to a coalition S is defined as $f(\mathbf{x}_S \cup \{x_i\}) - f(\mathbf{x}_S)$. Now the goal is to find an attribution to each feature which satisfies the the following properties.

1. *Local Accuracy.* The attributions for features in $\mathbf{x}$ sum up to the model prediction $f(\mathbf{x})$.
2. *Missingness.* If $x_i = 0$, then the attribution $\phi(x_i, \mathbf{x}) = 0$.
3. *Consistency.* If a model changes so that the marginal contributions of feature i to any subsets of players S increase or stay the same, then the attribution $\phi(x_i, \mathbf{x})$ do not increase. Formally, if for any subset of players S, $f'(\mathbf{x}_S \cup \{x_i\}) - f'(\mathbf{x}_S) \geq f(\mathbf{x}_S \cup \{x_i\}) - f(\mathbf{x}_S)$ holds, then $\phi(x_i, \mathbf{x}_S)$ for f' should not be less than f.

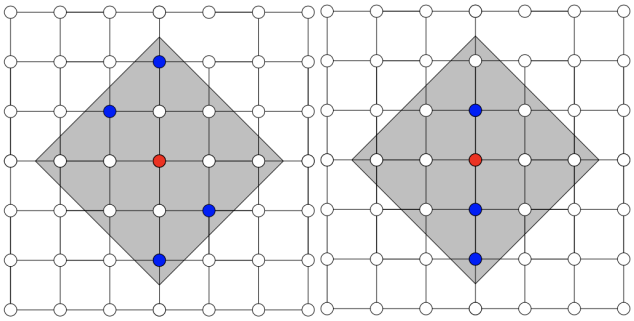
There is an attribution that uniquely satisfies these properties, known as Shapley Values (Shapley, 1952). The attribution is written as,

$$\phi(x_i, \mathbf{x}) = \sum_{S \subseteq G \setminus i} \frac{|S|!(|G| - |S| - 1)!}{|G|!} [f(x_{S \cup \{i\}}) - f(x_S)] \tag{2}$$

To make this weight term more intuitive, we rewrite it as $\frac{|S|!(|G|-|S|-1)!}{|G|!} = (|G| C_{|G|-1}^{|S|})^{-1}$. We see the Shapley value can be interpreted as

$$\phi(x_i, \mathbf{x}) = \frac{1}{\text{Number of features}} \sum_{\text{Coalition excluding i}} \frac{\text{Marginal contribution of i to the coalition}}{\text{Number of coalitions excluding i of this size}} \tag{3}$$

Unfortunately, the computational cost for Shapley values is exponential to the number of features, as it sums over all the subsets of input features. SHAP modify existing additive attribution algorithms, such as LIME, to make them an efficient approximation of Shapley values.



(a) L-Shapley (b) C-Shapley

Figure 3: Illustration of L-Shapley and C-Shapley in a 2-d grid (images). Each dot stands for a pixel. In this case, we compute the Shapley value attribution for the center pixel. The dark region is the neighborhood region specified by a range 2. The blue dots around the red dot is a drawn sample to compute marginal contribution for computing L-Shapley and C-Shapley respectively. Note that both algorithms only sample dots inside the neighborhood region, while C-Shapley further require the blue dots to be connected.

Kernel SHAP. From the uniqueness of the attribution, we know heuristically chosen sample weights by LIME break the properties listed above. However, by how each perturbed sample is weighted for fitting the local linear model, LIME could be a fast approximation of Shapeley value. The algorithm is known as Kernel SHAP.

Recall that LIME perturbs the input feature $\mathbf{x}$ to obtain a sample set. Let the perturbation be the random "masking" of some of the features in $\mathbf{x}$, and the remaining features as $\mathbf{x}_S$. The weight for x_S is $\frac{|G|-1}{C_{|G|}^{|S|}|S|(|G|-|S|)}$.

2.4 L-Shapley and C-Shapley for Structured Data

Still, Kernel SHAP requires drawing a lot of samples to get a good approximation of Shapley values. To further reduce the computational burden, Local Shapley (L-Shapley) and Connected Shapley (C-Shapley) (Chen et al., 2019) leverage assumptions on data distribution to approximate Shapley Values.

L-Shapley and C-Shapley work for structured input data. Both image and text data are structured: if we define a edge between neighboring pixels or neighboring words, we get a 2-d and 1-d grid respectively. Now, we plot each selection of a S to the grid. L-Shapley evaluate marginal contributions only for the dots lie in a neighboring region in the graph with a distance less or equal to k. C-Shapley further restrict the dots to be connected in a grid. See Figure 3 for illustrations.

The computational complexity of L-Shapley is only exponential to k, while the computational complexity of C-Shapley is further reduced to polynomial. They are be good approximations for Shapley values under some assumptions of data distributions. See original paper for more details.

2.5 Beyond Feature Attribution: Concept Based Explanations

Local feature attribution algorithms have some limitations: users have no control over what concepts these saliency maps pick up. For example, suppose we would like to figure out whether the the model make predictions for "zebras" by knowing that "zebras have strips" and that "zebras have four legs". Feature attribution methods fail to help in this case. A recent line of work on concept based explanations tackles this challenge. We will introduce Testing with Concept Activation (TCAV) algorithm in this lecture.

Suppose we already have a trained "zebra" classifier. To perform concept based explanations, we require a predefined set of concepts (e.g., striped, black) and a set of examples associated with each concept. The algorithm obtains a set of Concept Activation Vectors (CAV) by training a linear classifier to distinguish the intermediate activations of the examples that belong to a concept and that

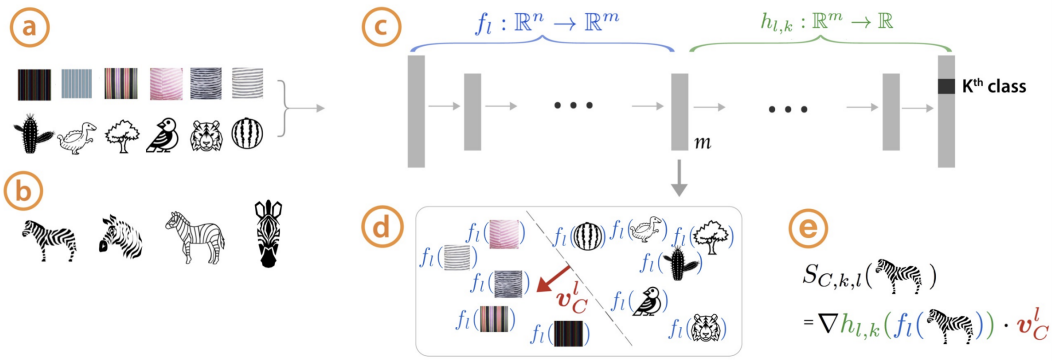


Figure 4: Testing with Concept Activation Vectors (TCAV) method for concept based explanations. Given a user-defined set of examples for a concept (e.g., 'striped'), and random examples (a), labeled training-data examples for the studied class (zebras) (b), and a trained network (c), TCAV can quantify the model's sensitivity to the concept for that class. CAVs are learned by training a linear classifier to distinguish between the activations produced by a concept's examples and examples in any layer (d) . The CAV is the vector orthogonal to the classification boundary. For the class of interest (zebras), TCAV uses the directional derivative $S_{C,k,l}(\mathbf{x})$ to quantify conceptual sensitivity (e) .

are chosen randomly. CAV is the vector orthogonal to the classification boundary. The CAV for class k at the l-th layer is written as v_c^l.

Now, we define *concept sensitivity* for an input image $\mathbf{x}$ as the prediction difference when the activation at the l-th moves slightly in the direction of the concept activation vector. Mathematically, we take the directional derivative at the l-th layer to the direction of the CAV.

$$S_{C,k,l} = \lim_{\epsilon \to 0} \frac{h_{l,k}(f_l(\mathbf{x}) + \epsilon \mathrm{v}_c^l) - h_{l,k}(f_l(\mathbf{x}))}{\epsilon} \tag{4}$$

TCAV for a class k regarding concept C at layer l is portion of example that have sensitivity greater than zero.

Recent study (Yeh et al., 2019) also explores automatic concept extraction for concept based explanations. Some follow-up research explore whether the concepts are complete to make predictions (Anonymous, 2020).

2.6 Global Explanations

In contrast to local explanation algorithms which explain a specific prediction, global explanation algorithms instead interpret global model behaviors, for example, which features are generally important to predict a class. Global explanations can be interpreted as rule extraction from trained neural networks.

The abovementioned TCAV is a global explanation. From the construction of TCAV, we know a quick way to obtain global explanations from local explanation algorithms is to average the local explanations over the dataset. Other global explanation approaches include various feature-level importance measures, such as conditioned expectation of predictions given that a feature exist.

3 Evaluating Explanation Algorithms

3.1 Is Visual Evaluation Reliable?

Most of the prior works employ qualitative analysis and human analysis as evaluation metrics. However, such evaluations can be unreliable. Figure 5 shows saliency maps provided by different explanation algorithms when the weights are *randomized* from the top prediction layer to the input layer. Surprisingly, for some explanation algorithms, the saliency maps still show the shape of the input image. Methods like Guided BP and Guided GradCAM give strikingly readable heatmaps when

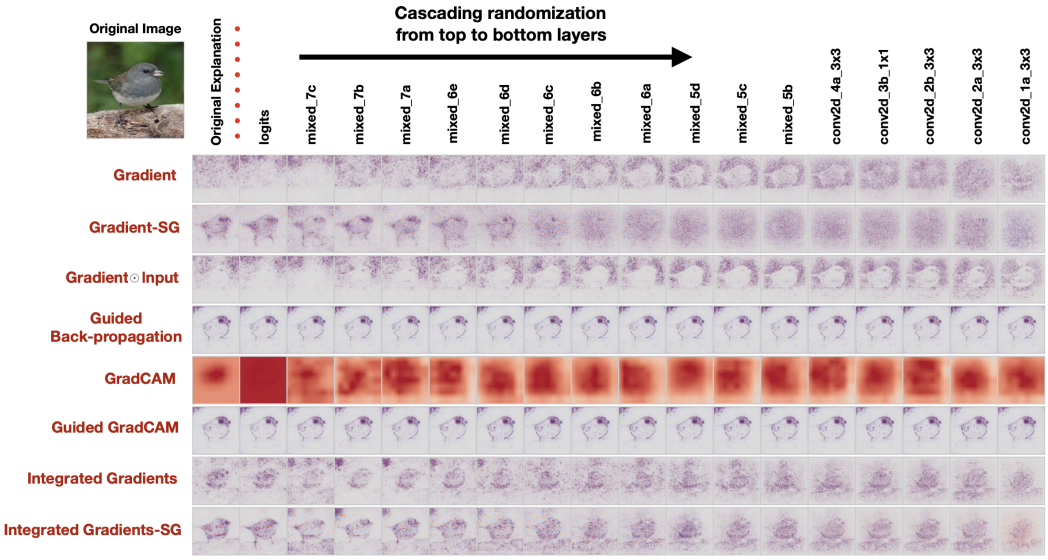


Figure 5: Saliency maps by different algorithms when we randomize the weights from top layers to bottom layers. Most of the saliency maps are still readable when the network is fully randomized.

the weights of the neural networks are fully randomized. In fact, theoretical analysis (Nie et al., 2018) show that these two algorithms are actually doing partial image recovery which is not related to the task of explanations.

The reasons for the failures are two fold. First, both Guided BP and Guided GradCAM introduce heuristics in the algorithms[1]. See (Nie et al., 2018) for the mathematical proofs that how it leads to perplexing behaviors of these algorithms. Secondly, it is because of the local connection of convolution neural networks: similar pixels will have similar activations even the weight is randomized. Therefore, the saliency maps are readable even when the weights are totally random.

In conclusion, (1) heuristically designed explanation algorithms can be unreliable. (2) Visual evaluation of saliency maps can be deceptive. To tackle the problem, it is necessary to check whether explanations are sensitive to classes, instead of behaving like a class-agnostic edge detector. We can also design various human evaluation paradigms, for example, evaluating whether users can identify a more accurate model from explanations (Singh et al., 2018).

3.2 Evaluating Explanation Algorithms with Probing Tasks

Poerner et al. (2018) propose two probing tasks for evaluating explanations and evaluate some popular explanation algorithms. The two probing tasks are namely the *hybrid document experiments* and the *morphosyntactic agreement experiments*.

1. *Hybrid document experiments* evaluate explanation algorithms on the text classification tasks. Given a target class c, we select a sentence from the class and randomly select N sentences from other classes. The $N+1$ sentences are randomly ordered and concatenated to feed into a trained text classifier. The evaluate counts a success for an explanation algorithm if the word with the top attribution score lies in the sentence of class c.
2. *Morphosyntactic agreement experiments*. The model is trained to predict whether a verb should be in the plural or singular form. For example, the model should predict "are" for the sentence "the children with the telescope (is/are) home". Now, the evaluation tests whether the word with the top attribution score is indeed the correct noun. For example, in the example above, the explanation algorithm is supposed to attribute the top importance score for the word "children".

[1]More specifically, both algorithms back propagate relevance scores from the output to the input by layers. Unlike gradients, *guided* GradCAM and BP treat ReLU specially: they allow backproping positive relevance scores through ReLU when the input of the ReLU ≤ 0.

The experiments conclude that empirically DeepLIFT and LIME perform better comparing to input occlusion and Integrated Gradients.

3.3 How NOT to Evaluate Explanation Algorithms

A good way to check an explanation algorithms is to take an *axiomatic* approach (Sundararajan et al., 2017) (Lundberg & Lee, 2017) at the first place, i.e., predefine a set of properties that our explanations ought to satisfy. If the designed algorithms pass the theoretical check of these properties, we guarantee that the algorithm reliably explains some properties about models.

References

Anonymous. On concept-based explanations in deep neural networks. In *Submitted to International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BylWYC4KwH`. under review.

Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1MXz20cYQ`.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. L-shapley and c-shapley: Efficient model interpretation for structured data. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1E3Ko09F7`.

Sarthak Jain and Byron C Wallace. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3543–3556, 2019.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda B. Viégas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *ICML*, 2017.

Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016.

Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *ArXiv*, abs/1705.07874, 2017.

Christoph Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

Weili Nie, Yang Zhang, and Ankit Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. *arXiv preprint arXiv:1805.07039*, 2018.

Nina Poerner, Hinrich Schütze, and Benjamin Roth. Evaluating neural network explanation methods using hybrid documents and morphological agreement. *arXiv preprint arXiv:1801.06422*, 2018.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *HLT-NAACL Demos*, 2016.

LS Shapley. A value for n-person games. Technical report, RAND CORP SANTA MONICA CA, 1952.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *ArXiv*, abs/1704.02685, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Chandan Singh, W James Murdoch, and Bin Yu. Hierarchical interpretations for neural network predictions. *arXiv preprint arXiv:1806.05337*, 2018.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, 2017.

Chih-Kuan Yeh, Been Kim, Sercan O Arik, Chun-Liang Li, Pradeep Ravikumar, and Tomas Pfister. On concept-based explanations in deep neural networks. *arXiv preprint arXiv:1910.07969*, 2019.

Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.