Part 2: Methods for Explaining DNNs

Wojciech Samek, Grégoire Montavon

September 18, 2020

Outline of Part 2

- Defining the problem of explanation
- Self-explainable models
  - Advantages & limitations
- Post-hoc explanations
  - Perturbation-based approaches
  - Propagation-based approaches

Defining The Problem of Explanation

- Consider we have a trained model f.
- We give to this model a data point $\mathbf{x} \in \mathbb{R}^d$, where each feature x_i composing it is assumed to be interpretable (e.g. physical measurement, pixel, or word).
- The model produces for $\mathbf{x}$ an output $f(\mathbf{x})$.
- We would like to build an explanation $\mathbf{R} = (R_i)_i$ indicating to what extent each feature i contributes to the prediction.

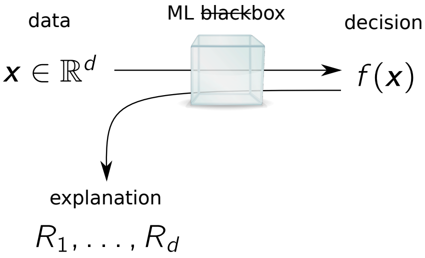


Illustration for a Linear Model

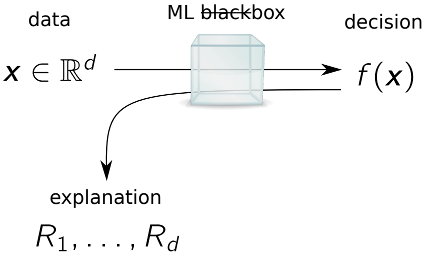
- **First step:** Compute the prediction

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}^\top \mathbf{x} \\
&= w_1 x_1 + w_2 x_2 + \cdots + w_d x_d
\end{aligned}
$$

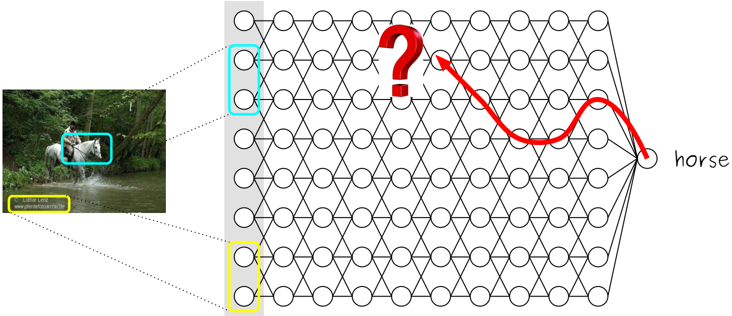
- **Second step:** Extract an explanation

$$
\begin{aligned}
R_1 &\leftarrow w_1 x_1 \\
R_2 &\leftarrow w_2 x_2 \\
&\vdots \\
R_d &\leftarrow w_d x_d
\end{aligned}
$$

$$
\mathbf{R} \leftarrow (R_1, R_2, \ldots, R_d)
$$

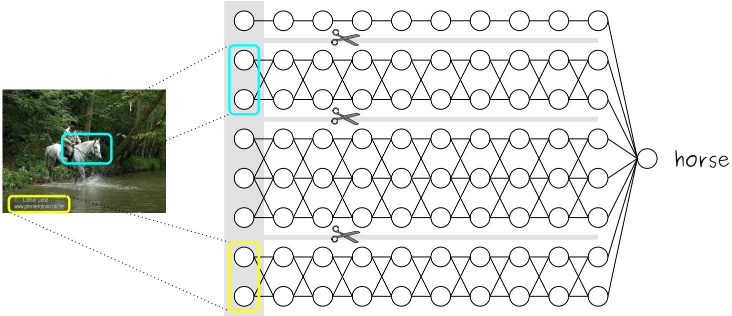


From Linear Models to Deep Networks



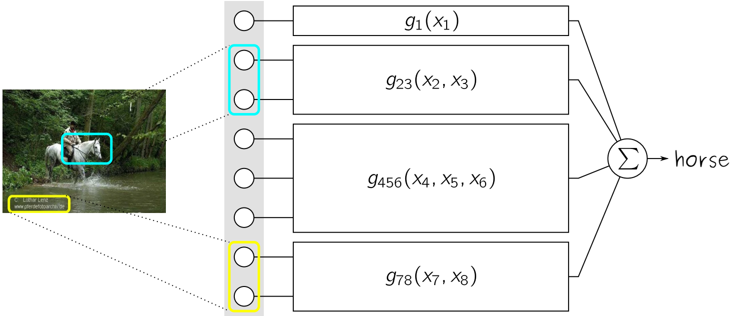
Question: How to trace which input features have contributed to the prediction in a more general deep model?

Self-Explainable Deep Networks



Idea: Restrict connectivity to ease the problem of attribution.

The Generalized Additive Model (GAM) [6]



Observation: Attribution is easy: $R_1 = g_1(x_1)$, $R_{23} = g_{23}(x_2, x_3)$ ...

Bag-Of-Local-Features [5]

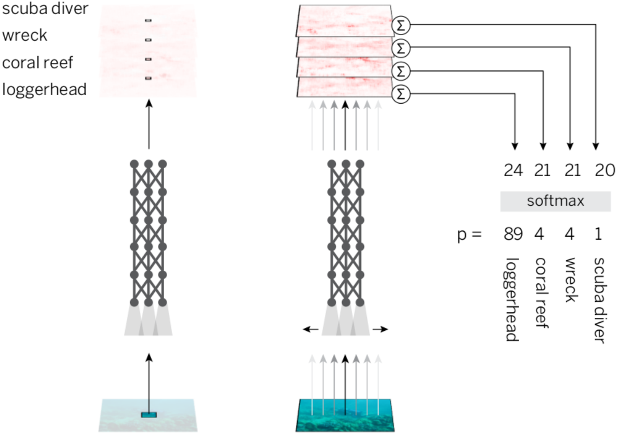
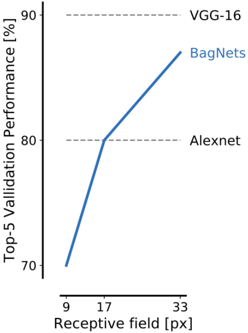


Image source:
Brendel et al. (2019) Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet

Bag-Of-Local-Features [5]



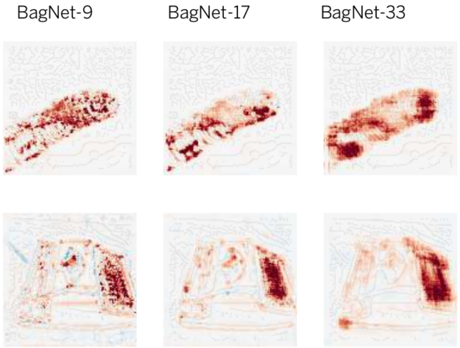


Image source:
Brendel et al. (2019) Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet

- With a larger receptive field (i.e. with less restrictions on the model), the prediction accuracy improves but the explanation becomes more blurry.

Advantages and Limitations of Self-Explainable Models

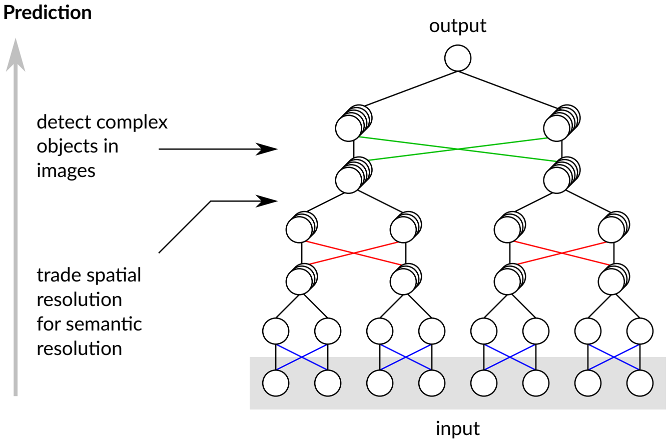
Advantages

- Explanations can be *easily* extracted without further analysis.
- The model can be designed to be *maximally interpretable* (e.g. by penalizing the use of uninterpretable features).
- Model constraints can be *relaxed* when explanation is coarse-grained (e.g. pixels $\rightarrow$ patches).

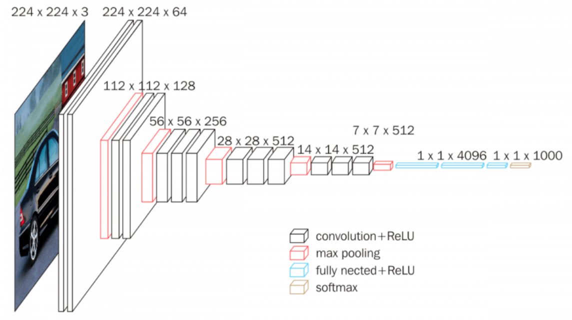
Limitations

- Self-explainable model might *lack representation power*, e.g. the GAM cannot represent a simple max-pooling operation.
- Even when the model predicts well ...
  - The model's strategy may be influenced by its restricted structure, and this may lead to a *less natural* prediction strategy from which it is harder to extract knowledge.
  - The model's strategy will likely be *computationally less efficient* than a standard model.

Beyond Generalized Additive Models



Example: Convolutional Neural Networks



Properties

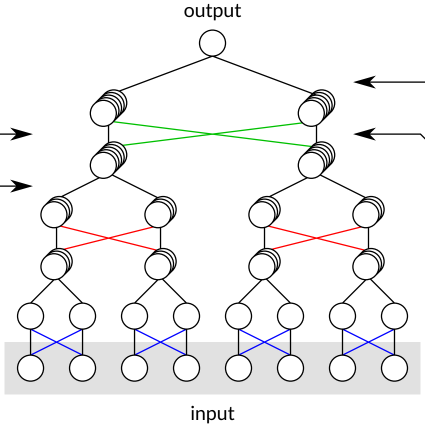
- Top-layers can capture long-range interactions.
- Increasingly many features can be built in higher layers.
- Representation remains finite-dimensional at each layer ($\rightarrow$ computationally efficient).

Explaining Beyond Generalized Additive Models

Prediction

detect complex objects in images

trade spatial resolution for semantic resolution

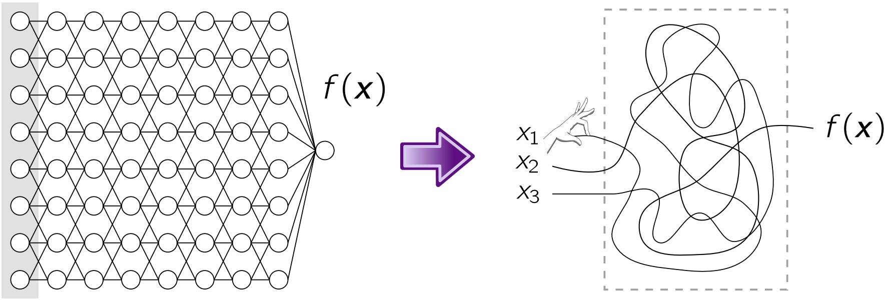


Explanation

spatial resolution too low for explanation

how to redistribute to pixels?

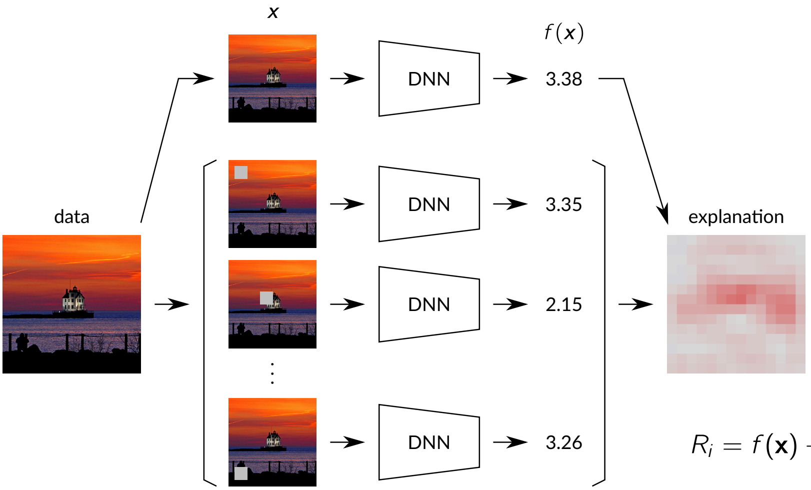
A Different Approach to Explanation: Perturbation



Examples from the literature:

- Occlusion [18], Prediction Difference Analysis [19]

Perturbation Analysis



$$R_i = f(\mathbf{x}) - f(\mathbf{x}_{-i})$$

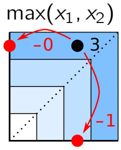
Perturbation Analysis

Advantages

- Can be applied to *any* function $f(\mathbf{x})$.
- Consistent for GAMs ($R_i = f(\mathbf{x}) - f(\mathbf{x}_{-i}) = g_i(\mathbf{x})$).

Limitations

- Slow (function f must be reevaluated for each occlusion)
- Intrinsically local, e.g. fails to explain max-pooling when several features in the pool are activated.
- Potentially biased by what is inserted in place of the removed patch. (Alternative: remove and inpaint [1, 13].)



Continuous Perturbations

- Consider a sequence of inputs $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$ interpolating between $\mathbf{x}^{(0)} = \mathbf{0}$ and $\mathbf{x}^{(N)} = \mathbf{x}$.
- Perform for each n the perturbation analysis

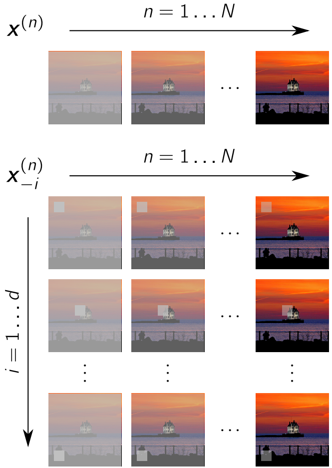
$$R_i^{(n)} = f(\mathbf{x}^{(n)}) - f(\mathbf{x}_{-i}^{(n)})$$

where

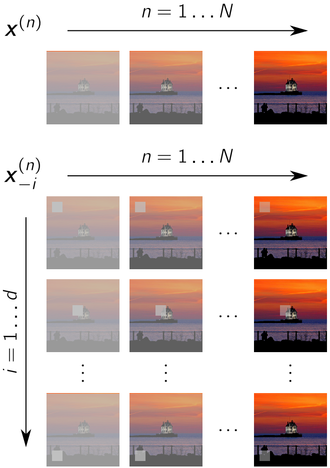
$$\mathbf{x}_{-i}^{(n)} = (x_1^{(n)}, \dots, x_{i-1}^{(n)}, x_i^{(n-1)}, x_{i+1}^{(n)}, \dots, x_d^{(n)})$$

- Sum them up:

$$R_i = \textstyle\sum_{n=1}^{N} R_i^{(n)}$$



Continuous Perturbations



- **Observation:** When the interpolation steps are small enough and when f is differentiable,

$$R_i^{(n)} \approx [\nabla f(\mathbf{x}^{(n)})]_i \cdot (\mathbf{x}_i^{(n)} - \mathbf{x}_i^{(n-1)})$$

  where the function's gradient appears.
- At each step, the perturbation for *all* dimensions can be computed using only one gradient evaluation.
- This is the integrated gradients method (in discretized form) [17].

Integrated Gradients and Gradient $\times$ Input

- Integrated Gradients (IG) [17]:

$$R_i = \sum_{n=1}^{N} [\nabla f(\mathbf{x}^{(n)})]_i \cdot (x_i^{(n)} - x_i^{(n-1)})$$

- Gradient $\times$ Input (GI) [15, 2, 9]:

$$R_i = [\nabla f(\mathbf{x})]_i \cdot x_i$$

i.e. an input feature i contributes if it is present in the data ($x_i > 0$) and if the model reacts to it ($[\nabla f(\mathbf{x})]_i > 0$).

Proposition: When $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$ linearly interpolate between $\mathbf{x}^{(0)} = \mathbf{0}$ and $\mathbf{x}^{(N)} = \mathbf{x}$, and when f is positively homogeneous, i.e. $\forall_{t \geq 0} : \ f(t\mathbf{x}) = tf(\mathbf{x})$, then IG and GI produce the same result.

Integrated Gradients and Gradient $\times$ Input

Proposition: When $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$ linearly interpolate between $\mathbf{x}^{(0)} = \mathbf{0}$ and $\mathbf{x}^{(N)} = \mathbf{x}$, and when f is positively homogeneous, i.e. $\forall_{t \geq 0} : \; f(t\mathbf{x}) = tf(\mathbf{x})$, then IG and GI produce the same result.

Proof: We start with IG and arrive at GI using a property of positively homogeneous functions (cf. note).

$$R_i = \sum_{n=1}^{N} [\nabla f(\mathbf{x}^{(n)})]_i \cdot (x_i^{(n)} - x_i^{(n-1)}) \qquad (1)$$

$$= \sum_{n=1}^{N} [\nabla f(\mathbf{x})]_i \cdot (x_i^{(n)} - x_i^{(n-1)}) \qquad (2)$$

$$= [\nabla f(\mathbf{x})]_i \cdot \sum_{n=1}^{N} (x_i^{(n)} - x_i^{(n-1)}) = [\nabla f(\mathbf{x})]_i \cdot x_i \qquad (3)$$

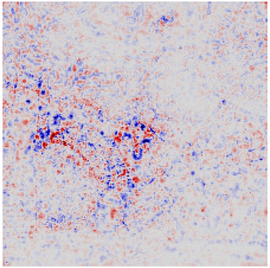
Note: A positively homogeneous function satisfies $\forall_{t \geq 0} : \; f(t\mathrm{x}) = tf(\mathrm{x})$. Differentiating on both sides gives

$$\frac{\partial}{\partial \mathrm{x}} f(t\mathrm{x}) = \frac{\partial}{\partial \mathrm{x}} tf(\mathrm{x})$$
$$t\nabla f(t\mathrm{x}) = t\nabla f(\mathrm{x})$$

therefore, the gradient is the same on any point on the segment $(0, \mathrm{x})$.

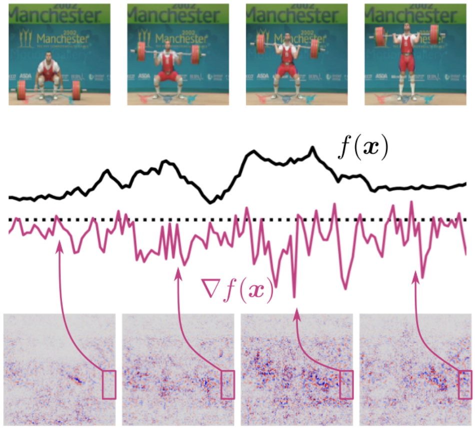
Gradient × Input in Practice

Example: Gradient × Input explanation of the VGG-16 neural network output neuron 'viaduct' for a given input image:



Observation: There is an exceedingly large amount of positive (red) and negative (blue) scores. Explanations also appear noisy and are hard to interpret.

Problem: Gradients are ‘Shattered’



- We look at the DNN output (and its gradient) along some trajectory in the input space, e.g. an athlete lifting a barebell.
- The function is relatively stable, but the gradient strongly oscillates and appears noisy (cf. [4]).

Shattered Gradients: A Construction

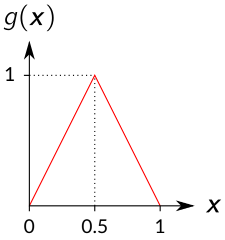
Consider the function:

$$g(x) = 2 \cdot \mathrm{ReLU}(x) - 4 \cdot \mathrm{ReLU}(x - 0.5)$$

defined on the interval $[0, 1]$.

We apply the function recursively to form a deep neural network.

function	output	max slope	# linear pieces
$g(x)$	$[0, 1]$	2	2
$g \circ g(x)$	$[0, 1]$	4	4
$g \circ g \circ g(x)$	$[0, 1]$	8	8
$g \circ g \circ g \circ g(x)$	$[0, 1]$	16	16



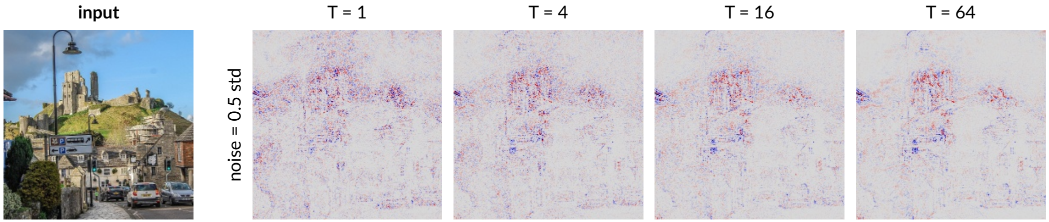
Potentially exponential growth of gradient and linear pieces (cf. [11]).

SmoothGrad [16]: "Removing Noise by Adding Noise"

Idea: Perform the gradient-based analysis with multiple random perturbations $\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_T$ of the input, and average the explanations.

Example: Smooth Gradient $\times$ Input

$$R_i = \frac{1}{T} \sum_{t=1}^{T} [\nabla f(\mathbf{x} + \boldsymbol{\epsilon}_t)]_i [\mathbf{x} + \boldsymbol{\epsilon}_t]_i$$



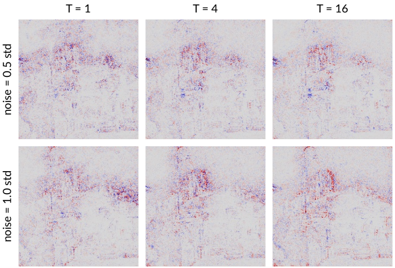
SmoothGrad

Advantages

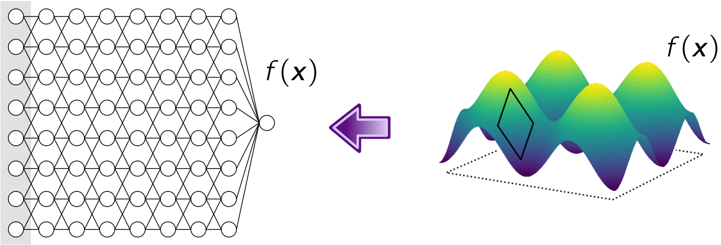
- Reduces explanation noise.
- Simple to implement (just call the same code multiple time)
- Widely applicable (can be applied on top of any explanation technique).

Limitations

- Computation cost increases by a factor T while explanation noise is in the best case only reduced by a factor $\sqrt{T}$.
- Adding noise to the input implies that we explain a slightly different quantity than the input (this may add a bias to the explanation).



From Function-Based to Propagation-Based



Questions:

- Can using the structure of the network *explicitly* (e.g. by running a special propagation pass) help to produce a better explanation?
- Can this approach reduce explanation noise *without* having to evaluate the function multiple times?

The 'Deconvolution' Method [18]

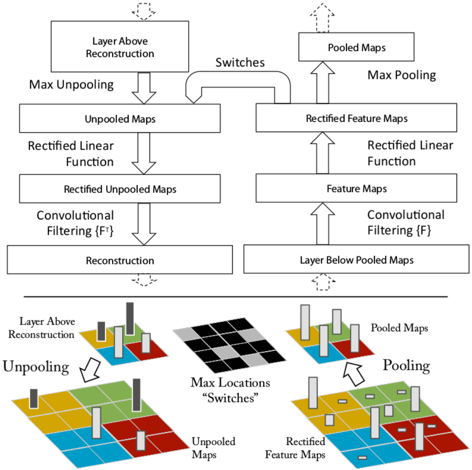


Image source:

Zeiler et al. (2014) Visualizing and Understanding Convolutional Networks

- Max-pooling layers: propagate to the winner
- Convolutional layers: convolve with transposed weights
- ReLU layers: apply the ReLU function

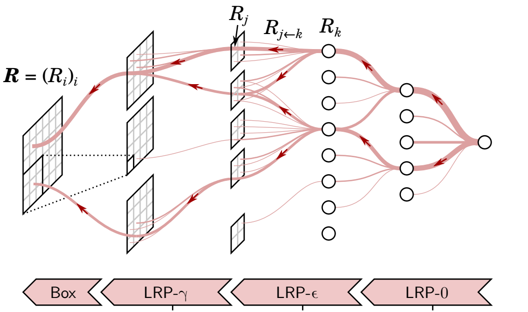
The ‘Deconvolution’ Method

Image source:

Zeiler et al. (2014) Visualizing and Understanding Convolutional Networks

- **Observation:** Gradient noise has disappeared $\Rightarrow$ leveraging structure is useful.
- **Limitation:** The method was meant as a visualization rather than as an explanation (it does not tell how much each input variable has contributed to the prediction).

Layer-wise Relevance Propagation (LRP) [3, 10]



$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

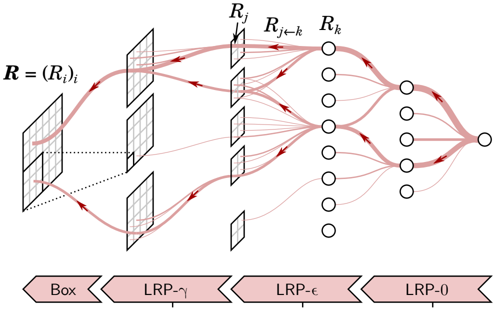
$$R_j = \sum_k \frac{a_j w_{jk}}{\varepsilon_k + \sum_{0,j} a_j w_{jk}} R_k$$

$$R_j = \sum_k \frac{a_j (w_{jk} + \gamma w_{jk}^{+})}{\sum_{0,j} a_j (w_{jk} + \gamma w_{jk}^{+})} R_k$$

Ideas:

- Use the structure of the neural network to robustly compute relevance scores for the input features.
- Propagate the output of the network backwards by means of propagation rules.
- Propagation rules can be tuned for explanation quality. E.g. sensitive in top-layers, robust in lower layers.

Layer-wise Relevance Propagation (LRP) [3, 10]



$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

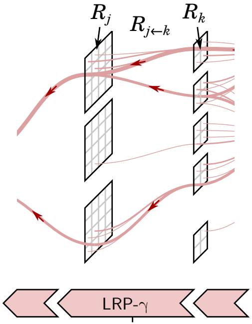
$$R_j = \sum_k \frac{a_j w_{jk}}{\varepsilon_k + \sum_{0,j} a_j w_{jk}} R_k$$

$$R_j = \sum_k \frac{a_j (w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j (w_{jk} + \gamma w_{jk}^+)} R_k$$

Some notation:

- j and k: neurons from successive layers
- w_{jk}: weight connecting neuron j to neuron k
- w_{0k}: bias for neuron k.
- $\sum_{0,j}$ sum over all input neurons j of neuron k and the bias.
- ReLU neuron: $a_k = \max\left(0, \sum_{0,j} a_j w_{jk}\right)$.

Dissecting a LRP Propagation Rule



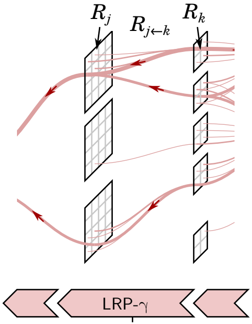
$$R_j = \sum_k \frac{a_j(w_{jk}+\gamma w_{jk}^+)}{\sum_{0,j} a_j(w_{jk}+\gamma w_{jk}^+)} R_k$$

Example: LRP-γ [10]

$$R_j = \sum_k \frac{a_j(w_{jk}+\gamma w_{jk}^+)}{\sum_{0,j} a_j(w_{jk}+\gamma w_{jk}^+)} R_k$$

- $a_j(w_{jk}+\gamma w_{jk}^+)$: Contribution of neuron a_j to the activation a_k.
- R_k 'Relevance' of neuron k available for redistribution.
- $\sum_{0,j} a_j(w_{jk}+\gamma w_{jk}^+)$ Normalization term that implements conservation.
- $\sum_k$: Pool all 'relevance' received by neuron j from the layer above.

Dissecting a LRP Propagation Rule (2nd view)



$$R_j = \sum_k \frac{a_j(w_{jk}+\gamma w_{jk}^+)}{\sum_{0,j} a_j(w_{jk}+\gamma w_{jk}^+)} R_k$$

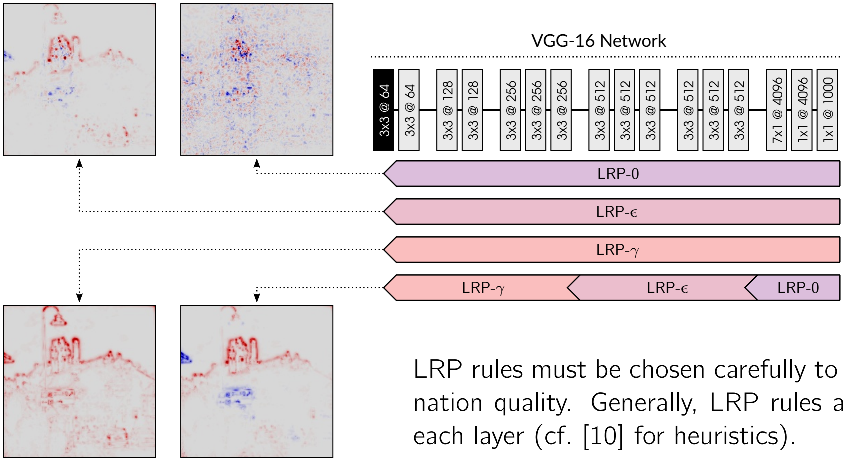
Example: LRP-γ [10]

$$R_j = a_j \cdot \Big(\sum_k \frac{(w_{jk}+\gamma w_{jk}^+)}{\sum_{0,j} a_j(w_{jk}+\gamma w_{jk}^+)} R_k \Big)$$

- a_j : Activation of neuron j.
- $\big(\sum_k \ldots\big)$: Sensitivity of neural network output to a_j.

i.e. similar interpretation as for Gradient × Input, but now at each layer.

Effect of LRP Rules on Explanation



LRP rules must be chosen carefully to deliver best explanation quality. Generally, LRP rules are set different at each layer (cf. [10] for heuristics).

Layer-Wise Relevance Propagation

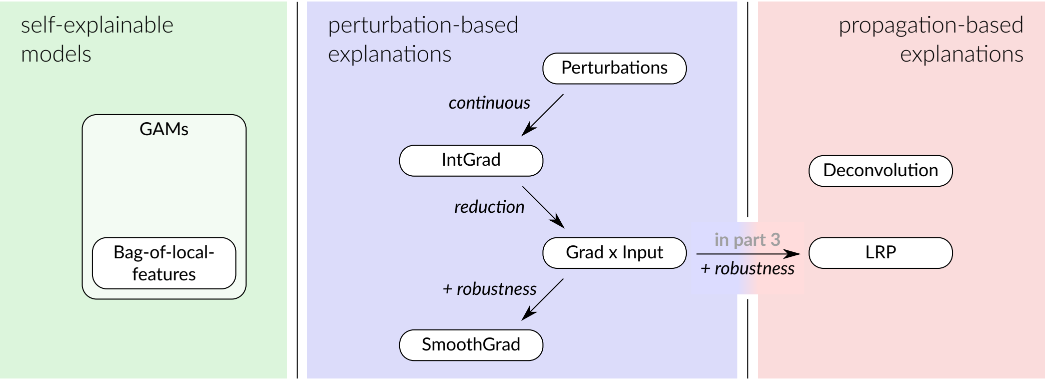
Advantages

- *Good explanation quality* on deep networks.
- *Fast* (in the order of a single forward/backward pass).
- *Flexible* (the multiple hyperparameters can be tuned to match the user needs).

Limitations

- The LRP propagation strategy must be adapted to each new architecture.
- LRP makes some assumptions about the structure of the model (i.e. it works for many neural networks but not for all models).

Connections between Explanation Methods



More Explanation Methods

Other methods that have been proposed to attribute the prediction to input features:

- LIME [12]: learns a local surrogate model and analyze it.
- SHAP [8]: based on the game theory framework of Shapley values.
- Meaningful Perturbations [7]: synthesizes an optimal perturbation with gradient ascent.
- Grad-CAM [14]: combines gradient-based and propagation-based approaches.

Summary of Part 2

- **Self-explainable models** can be practical, but they often lack sufficient representation power and can be computationally costly.
- Explaining general DNNs is hard (no directly identifiable contributions, gradient noise), but possible.
- Two important categories of 'post-hoc' explanation techniques (**perturbation-based** and **propagation-based**).
- The LRP explanation technique is specially designed to explain deep networks (perform attribution by taking advantage of the layered structure).

References I

[1] C. Agarwal, D. Schonfeld, and A. Nguyen.
Removing input features via a generative model to explain their attributions to classifier's decisions.
CoRR, abs/1910.04256, 2019.

[2] M. Ancona, E. Ceolini, C. Öztireli, and M. H. Gross.
Gradient-based attribution methods.
In *Explainable AI*, volume 11700 of *Lecture Notes in Computer Science*, pages 169–191. Springer, 2019.

[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek.
On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation.
PLoS ONE, 10(7):e0130140, 07 2015.

[4] D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W. Ma, and B. McWilliams.
The shattered gradients problem: If resnets are the answer, then what is the question?
In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 342–350. PMLR, 2017.

[5] W. Brendel and M. Bethge.
Approximating cnns with bag-of-local-features models works surprisingly well on imagenet.
In *ICLR (Poster)*. OpenReview.net, 2019.

References II

[6] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad.
Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission.
In *KDD*, pages 1721–1730. ACM, 2015.

[7] R. C. Fong and A. Vedaldi.
Interpretable explanations of black boxes by meaningful perturbation.
In *ICCV*, pages 3449–3457. IEEE Computer Society, 2017.

[8] S. M. Lundberg and S. Lee.
A unified approach to interpreting model predictions.
In *NIPS*, pages 4765–4774, 2017.

[9] G. Montavon.
Gradient-based vs. propagation-based explanations: An axiomatic comparison.
In *Explainable AI*, volume 11700 of *Lecture Notes in Computer Science*, pages 253–265. Springer, 2019.

[10] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller.
Layer-wise relevance propagation: An overview.
In *Explainable AI*, volume 11700 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2019.

References III

[11] G. F. Montúfar, R. Pascanu, K. Cho, and Y. Bengio.
On the number of linear regions of deep neural networks.
In *NIPS*, pages 2924–2932, 2014.

[12] M. T. Ribeiro, S. Singh, and C. Guestrin.
"why should I trust you?": Explaining the predictions of any classifier.
In *KDD*, pages 1135–1144. ACM, 2016.

[13] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller.
Toward interpretable machine learning: Transparent deep neural networks and beyond.
CoRR, abs/2003.07631, 2020.

[14] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra.
Grad-cam: Visual explanations from deep networks via gradient-based localization.
In *ICCV*, pages 618–626. IEEE Computer Society, 2017.

[15] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje.
Not just a black box: Learning important features through propagating activation differences.
CoRR, abs/1605.01713, 2016.

References IV

[16] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg.
Smoothgrad: removing noise by adding noise.
CoRR, abs/1706.03825, 2017.

[17] M. Sundararajan, A. Taly, and Q. Yan.
Axiomatic attribution for deep networks.
In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[18] M. D. Zeiler and R. Fergus.
Visualizing and understanding convolutional networks.
In *ECCV (1)*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014.

[19] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling.
Visualizing deep neural network decisions: Prediction difference analysis.
In *ICLR (Poster)*. OpenReview.net, 2017.