ICASSP 2017

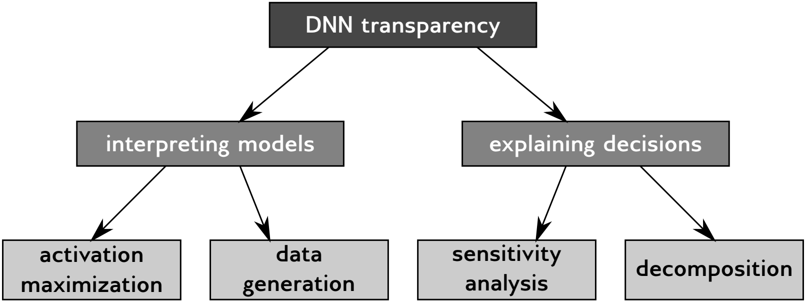
Tutorial on Methods for Interpreting and Understanding Deep Neural Networks

G. Montavon, W. Samek, K.-R. Müller

Part 2: Making Deep Neural Networks Transparent

5 March 2017

Making Deep Neural Nets Transparent



activation maximization
- Berkes 2006
- Erhan 2010
- Simonyan 2013
- Nguyen 2015/16

data generation
- Hinton 2006
- Goodfellow 2014
- v. den Oord 2016
- Nguyen 2016

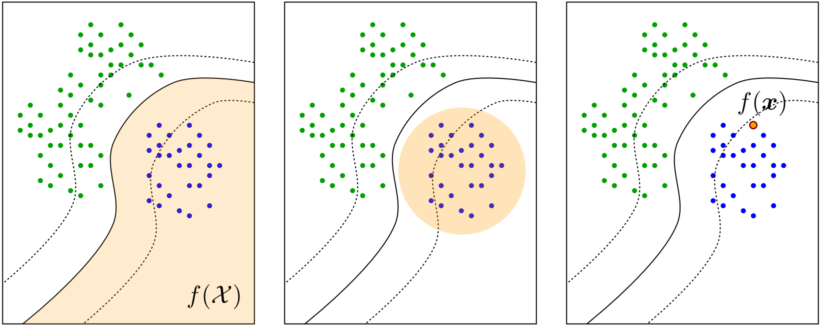
sensitivity analysis
- Khan 2001
- Gevrey 2003
- Baehrens 2010
- Simonyan 2013

decomposition
- Poulin 2006
- Landecker 2013
- Bach 2015
- Montavon 2017

focus on model ⟷ *focus on data*

Making Deep Neural Nets Transparent

model analysis → **decision analysis**



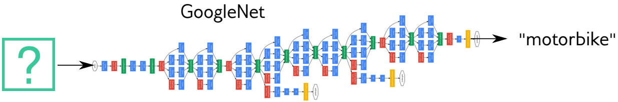
- visualizing filters
- max. class activation

- include distribution (RBM, DGN, etc.)

- sensitivity analysis
- decomposition

Interpreting Classes and Outputs

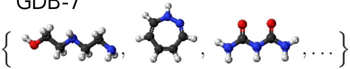
Image classification:

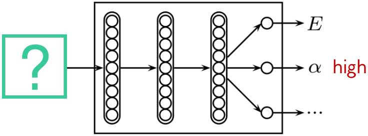


Question: How does a "motorbike" typically look like?

Quantum chemical calculations:

GDB-7

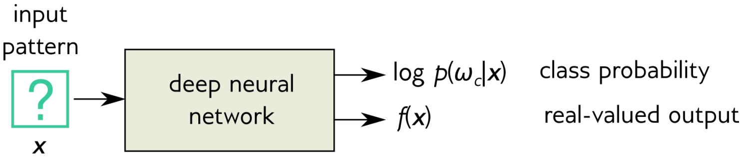




Question: How to interpret "α high" in terms of molecular geometry?

The Activation Maximization (AM) Method

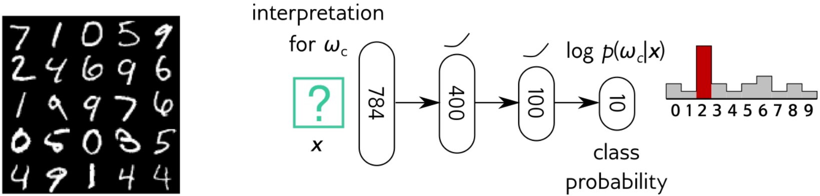
Let us interpret a concept predicted by a deep neural net (e.g. a class, or a real-valued quantity):

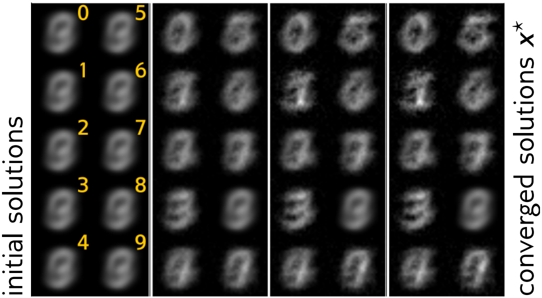


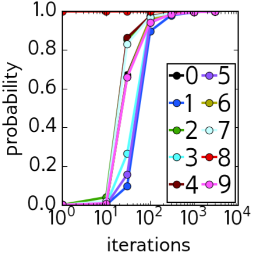
Examples:

- Creating a class prototype: $\max_{\boldsymbol{x} \in \mathcal{X}} \; \log p(\omega_c | \boldsymbol{x})$.
- Synthesizing an extreme case: $\max_{\boldsymbol{x} \in \mathcal{X}} \; f(\boldsymbol{x})$.

Interpreting a Handwritten Digits Classifier



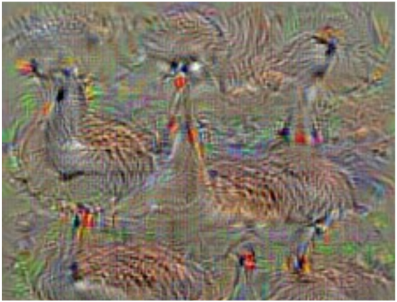




$\rightarrow \quad \rightarrow$ optimizing $\max_x \; p(\omega_c|x)$ $\rightarrow \quad \rightarrow$

Interpreting a DNN Image Classifier

goose



ostrich

Images from **Simonyan et al. 2013** "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps"

Observations:

- AM builds typical patterns for these classes (e.g. beaks, legs).
- Unrelated background objects are not present in the image.

Improving Activation Maximization

Activation-maximization produces class-related patterns, but they are not resembling true data points. This can lower the quality of the interpretation for the predicted class ω_c.

Idea:

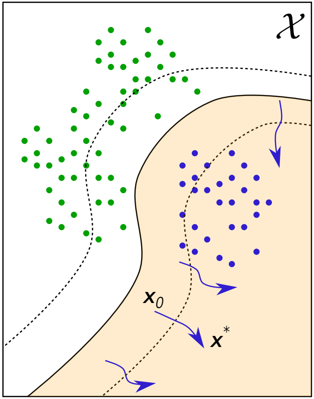
- Force the interpretation $\boldsymbol{x}^\star$ to match the data more closely.

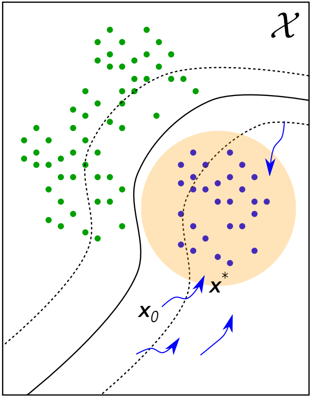
This can be achieved by redefining the optimization problem:

Find the input pattern that maximizes class probability.	$\rightarrow$	Find the most likely input pattern for a given class.

Improving Activation Maximization

Find the input pattern that maximizes class probability.	→	Find the most likely input pattern for a given class.





Improving Activation Maximization

Find the input pattern that maximizes class probability.	$\rightarrow$	Find the most likely input pattern for a given class.

Nguyen et al. 2016 introduced several enhancements for activation maximization:

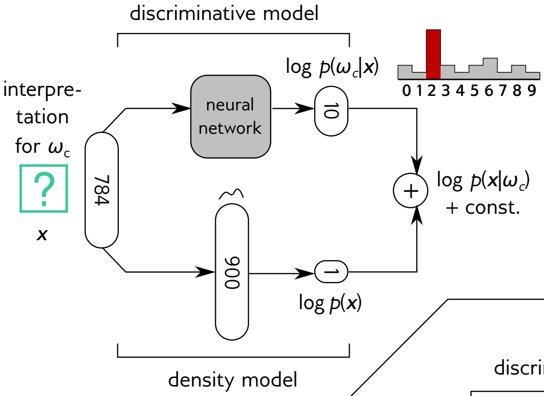
- Multiplying the objective by an expert $p(\boldsymbol{x})$:

$$p(\boldsymbol{x}|\omega_c) \propto \underbrace{p(\omega_c|\boldsymbol{x})}_{\text{old}} \cdot p(\boldsymbol{x})$$

- Optimization in code space:

$$\max_{\boldsymbol{z} \in \mathcal{Z}} \; p(\omega_c | \underbrace{g(\boldsymbol{z})}_{\boldsymbol{x}}) + \lambda \|\boldsymbol{z}\|^2 \qquad \boldsymbol{x}^\star = g(\boldsymbol{z}^\star)$$

These two techniques require an unsupervised model of the data, either a density model $p(\boldsymbol{x})$ or a generator $g(\boldsymbol{z})$.

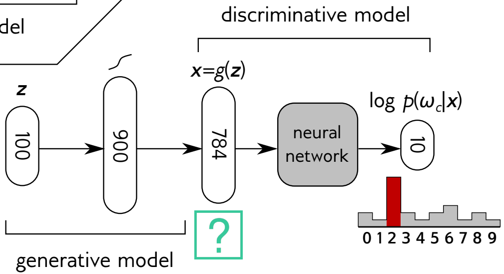


AM + density

- optimum has clear meaning
- objective can be hard to optimize

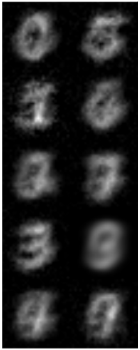
AM + generator

- more straightforward to optimize
- not optimizing log p($\boldsymbol{x}|\omega_c$)

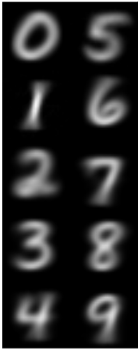


Comparison of Activation Maximization Variants

simple AM (initialized to mean)

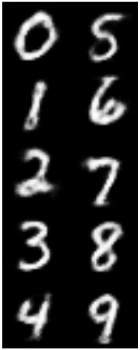


simple AM (init. to class means)



AM-density (init. to class means)

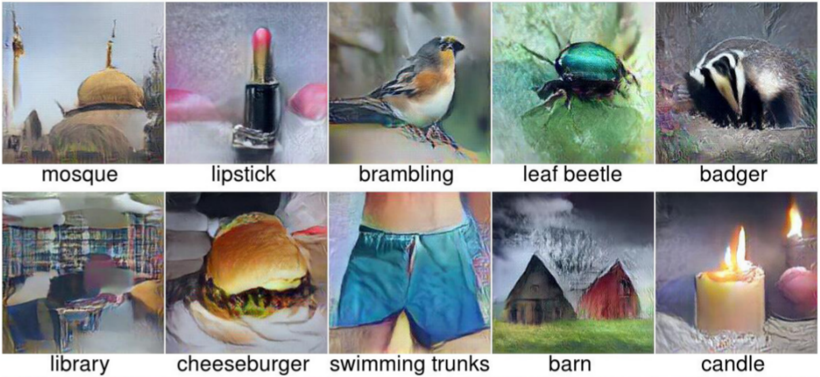
AM-gen (init. to class means)



Observation: Connecting to the data leads to sharper prototypes.

Enhanced AM on Natural Images

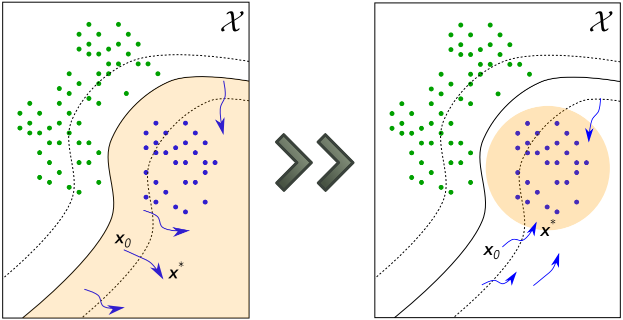
Images from **Nguyen et al. 2016.** "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks"



Observation: Connecting AM to the data distribution leads to more realistic and more interpretable images.

Summary

- Deep neural networks can be interpreted by finding input patterns that maximize a certain output quantity (e.g. class probability).
- Connecting to the data (e.g. by adding a generative or density model) improves the interpretability of the solution.



Limitations of Global Interpretations

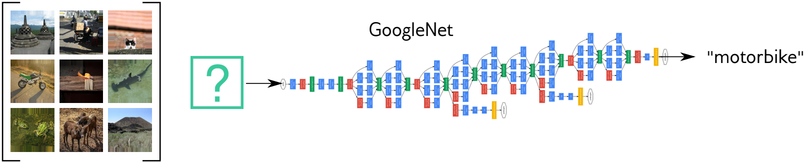
Question: Below are some images of motorbikes. What would be the best prototype to interpret the class "motorbike"?

Observations:

- Summarizing a concept or category like "motorbike" into a single image can be difficult (e.g. different views or colors).
- A good interpretation would grow as large as the diversity of the concept to interpret.

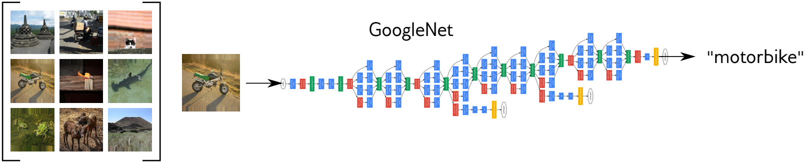
From Prototypes to Individual Explanations

Finding a prototype:



Question: How does a “motorbike” typically look like?

Individual explanation:



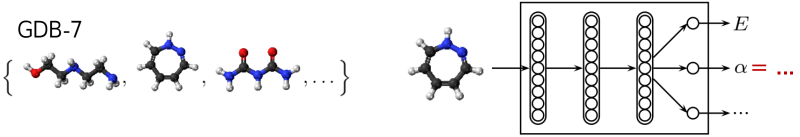
Question: Why is *this* example classified as a motorbike?

From Prototypes to Individual Explanations

Finding a prototype:

Question: How to interpret "α high" in terms of molecular geometry?

Individual explanation:

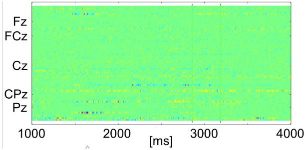


Question: Why α has a certain value for *this* molecule?

From Prototypes to Individual Explanations

Other examples where individual explanations are preferable to global interpretations:

- **Brain-computer interfaces:** Analyze input data for a *given* user at a *given* time in a *given* environment.

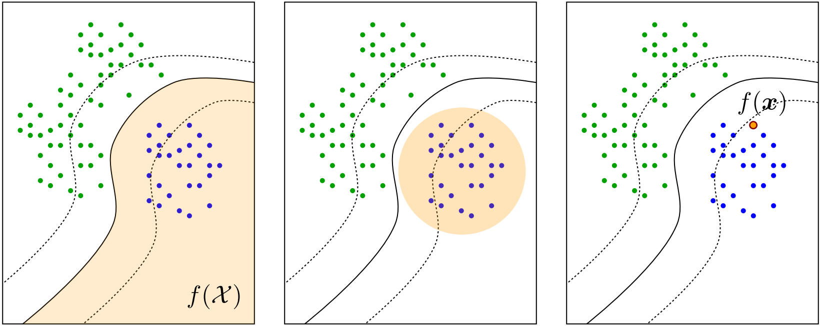


- **Personalized medicine:** Extracting the relevant information about a medical condition for a *given* patient at a *given* time.

Each case is unique and needs its own explanation.

From Prototypes to Individual Explanations

model analysis → **decision analysis**



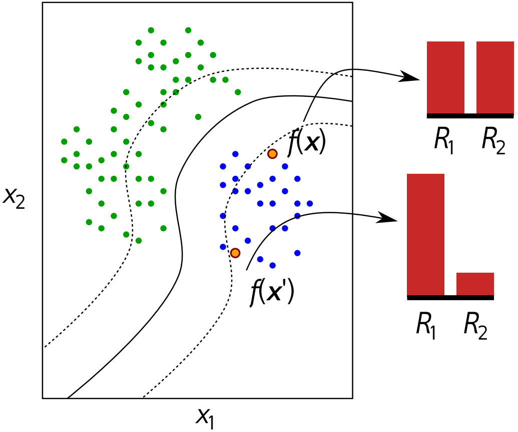
- visualizing filters
- max. class activation

- include distribution (RBM, DGN, etc.)

- sensitivity analysis
- decomposition

Explaining Decisions

Goal: Determine the relevance of each input variable for a given decision $f(x_1, x_2, \ldots, x_d)$, by assigning to these variables *relevance* scores $R_1, R_2, \ldots, R_d$.



Basic Technique: Sensitivity Analysis

Consider a function f, a data point $\boldsymbol{x} = (x_1, \ldots, x_d)$, and the prediction

$$f(x_1, \ldots, x_d).$$

Sensitivity analysis measures the local variation of the function along each input dimension

$$R_i = \left(\frac{\partial f}{\partial x_i} \Big|_{x=\boldsymbol{x}} \right)^2$$

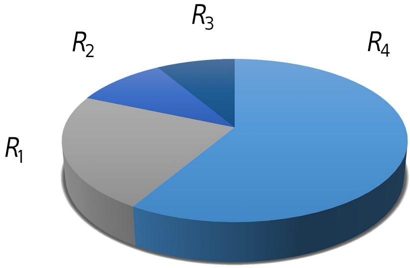
Remarks:

- Easy to implement (we only need access to the gradient of the decision function).
- But does it really explain the prediction?

Explaining by Decomposing

aggregate quantity

$f(\boldsymbol{x}) =$



decomposition

$$\boxed{\sum_i R_i = f(\boldsymbol{x})}$$

Examples:

- Economic activity (e.g. petroleum, cars, medicaments, ...)
- Energy production (e.g. coal, nuclear, hydraulic, ...)
- Evidence for object in an image (e.g. pixel 1, pixel 2, pixel 3, ...)
- Evidence for meaning in a text (e.g. word 1, word 2, word 3, ...)

What Does Sensitivity Analysis Decompose?

Sensitivity analysis

$$R_i = \left(\frac{\partial f}{\partial x_i} \Big|_{\boldsymbol{x}=\boldsymbol{x}} \right)^2$$

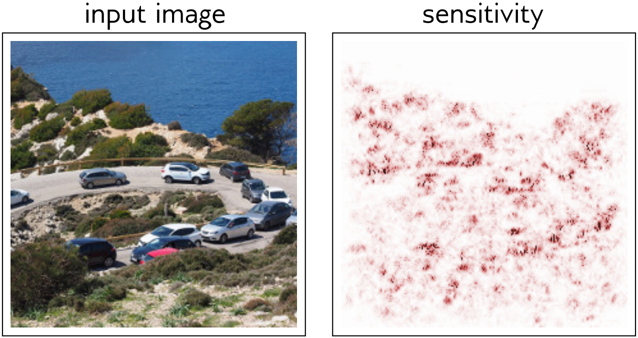
is a decomposition of the gradient norm $\|\nabla_{\boldsymbol{x}} f\|^2$.

Proof: $\sum_i R_i = \|\nabla_{\boldsymbol{x}} f\|^2$ $\square$

Sensitivity analysis explains a *variation* of the function, not the function value itself.

What Does Sensitivity Analysis Decompose?

Example: Sensitivity for class "car"



- Relevant pixels are found both on cars and on the background.
- Explains what *reduces/increases* the evidence for cars rather what *is* the evidence for cars.

Decomposing the Correct Quantity

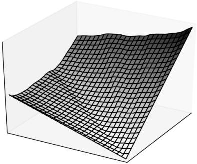
slope decomposition $\boxed{\sum_i R_i = \|\nabla_x f\|^2}$ $\rightarrow$ value decomposition $\boxed{\sum_i R_i = f(\boldsymbol{x})}$

Candidate: Taylor decomposition

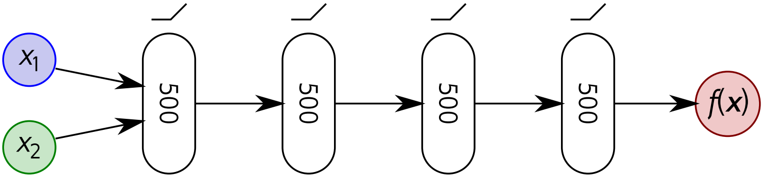
$$f(\boldsymbol{x}) = \underbrace{f(\widetilde{\boldsymbol{x}})}_{0} + \sum_{i=1}^{d} \underbrace{\frac{\partial f}{\partial x_i}\Big|_{\boldsymbol{x}=\widetilde{\boldsymbol{x}}} (x_i - \widetilde{x}_i)}_{R_i} + \underbrace{O(\boldsymbol{x}\boldsymbol{x}^\top)}_{0}$$

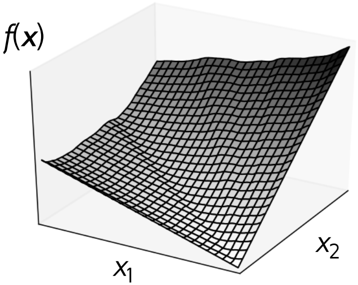
- Achievable for linear models and deep ReLU networks without biases, by choosing:

$$\widetilde{\boldsymbol{x}} = \lim_{\varepsilon \to 0} \varepsilon \cdot \boldsymbol{x} \approx \boldsymbol{0}.$$

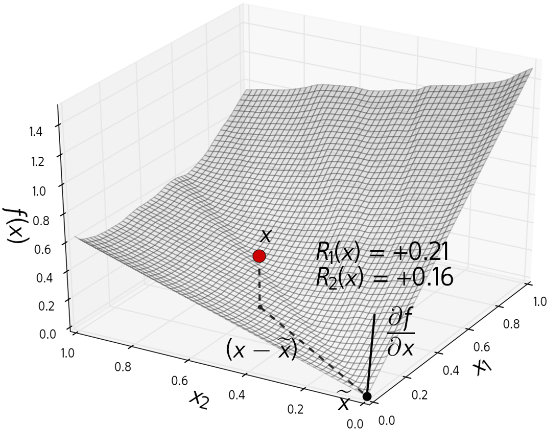


Experiment on a Randomly Initialized DNN



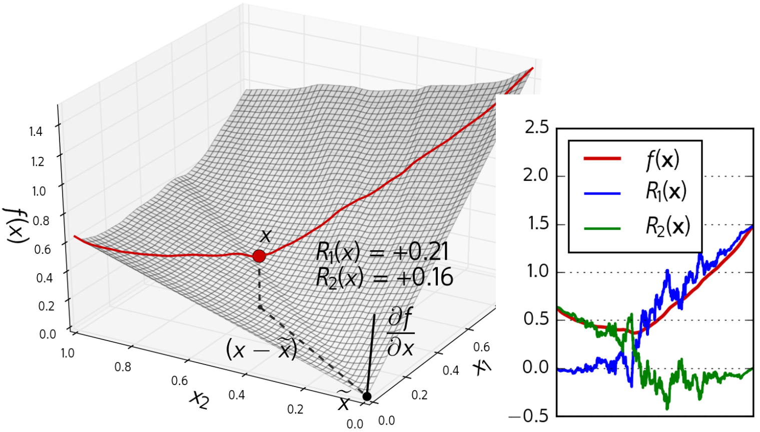


Decomposing the Output of the DNN



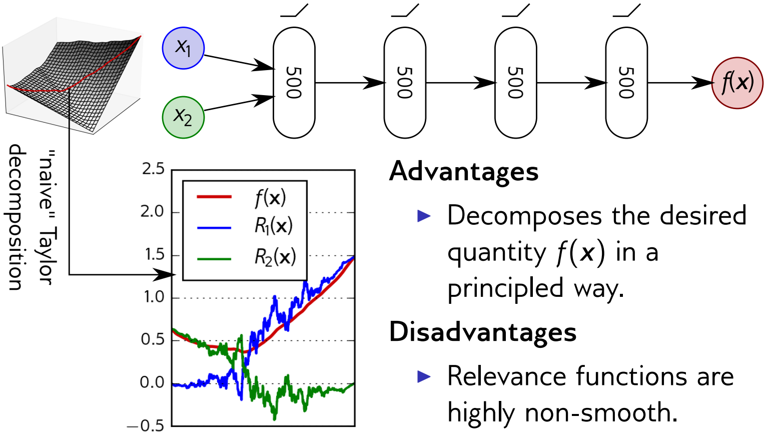
$$\boxed{R_i = \frac{\partial f}{\partial x_i}\Big|_{\boldsymbol{x}=\widetilde{\boldsymbol{x}}} \cdot (x_i - \widetilde{x}_i)}$$

Decomposing the Output of the DNN



$$\boxed{R_i = \frac{\partial f}{\partial x_i}\Big|_{\mathbf{x}=\widetilde{\mathbf{x}}} \cdot (x_i - \widetilde{x}_i)}$$ $\Rightarrow$ **"Naive" Taylor decomposition**

Decomposing the Output of the DNN



Advantages

- Decomposes the desired quantity $f(\boldsymbol{x})$ in a principled way.

Disadvantages

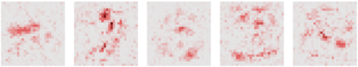
- Relevance functions are highly non-smooth.
- Relevance scores are sometimes negative.
- Inflexible w.r.t. the model.

Experiment on Handwritten Digits

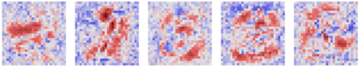
Data to classify:

3-layer MLP:

Sensitivity analysis



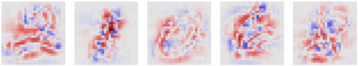
Naive Taylor ($\widetilde{\boldsymbol{x}} = 0$)



6-layer CNN:

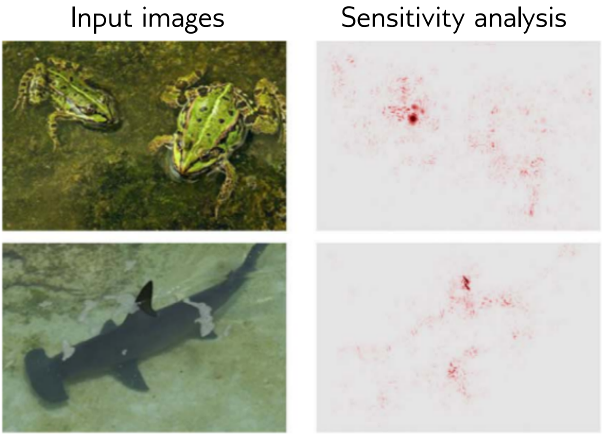
Sensitivity analysis

Naive Taylor ($\widetilde{\boldsymbol{x}} = 0$)



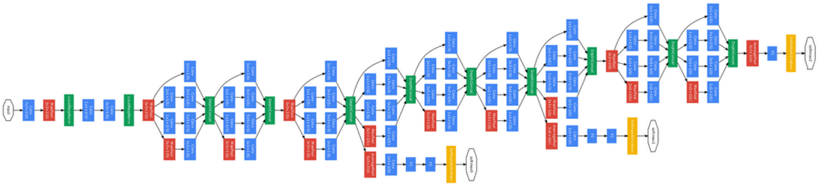
Observation: Both analyses produce noisy explanations of the MLP and CNN predictions.

Experiment on BVLC CaffeNet



Observation: Explanations are noisy and (over/under)represent certain regions of the image.

Explaining DNN Predictions

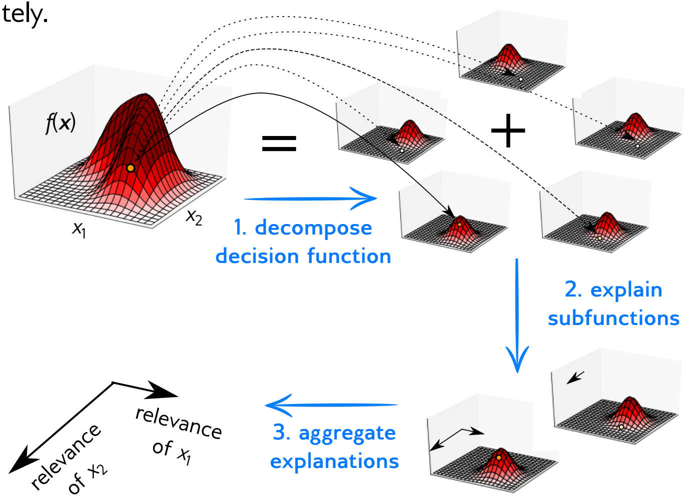


- Standard methods (sensitivity analysis, naive Taylor decomposition) are subject to gradient noise and do not work well on deep neural networks.

DNN predictions need more advanced explanation methods.

From Shallow to Deep Explanations

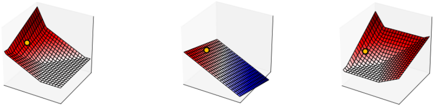
Key Idea: If a decision is too complex to explain, break the decision function into sub-functions, and explain each sub-decision separately.



From Shallow to Deep Taylor Decomposition

Taylor decomposition (TD)

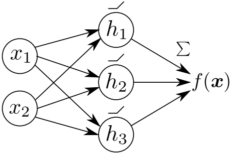
$$f(\boldsymbol{x}), \nabla f, \dots$$

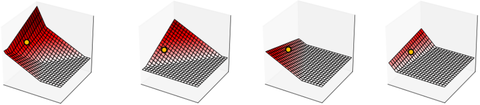


$$f(\boldsymbol{x}) = \nabla f\big|_{\boldsymbol{x}=\widetilde{\boldsymbol{x}}}^{\top} \cdot (\boldsymbol{x} - \widetilde{\boldsymbol{x}}) + \varepsilon$$

$$f(\boldsymbol{x}) = R_1 + R_2 + \varepsilon$$

deep Taylor decomposition (DTD)





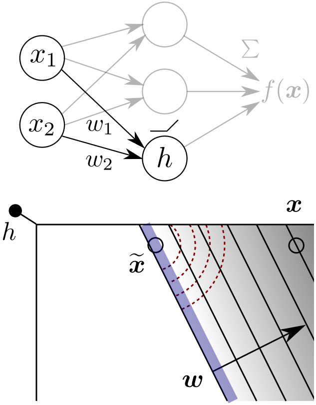
$$f(\boldsymbol{x}) = h_1 + h_2 + h_3$$

TD TD TD

Σ Σ

$$f(\boldsymbol{x}) = R_1 + R_2$$

Decomposing a Single Neuron



Equation of the ReLU neuron

$$h = \max(0, \boldsymbol{x}^\top \boldsymbol{w} + b)$$

Pick an appropriate root point

$$\widetilde{\boldsymbol{x}} \in \{\boldsymbol{x} : h \approx 0 \wedge \text{constraints}\}$$

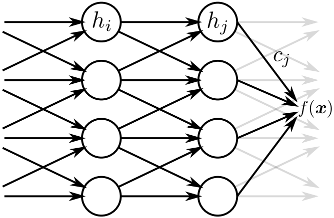
Perform a Taylor expansion and identify first-order terms

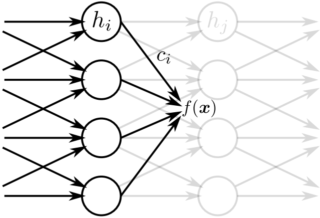
$$h = \nabla h\big|_{\widetilde{\boldsymbol{x}}}^\top \cdot (\boldsymbol{x} - \widetilde{\boldsymbol{x}}) = \sum_i \underbrace{w_i \cdot (x_i - \widetilde{x}_i)}_{R_i}$$

Resulting decomposition for various $\widetilde{\boldsymbol{x}}$

$$\underbrace{R_i = \frac{x_i w_i^+}{\sum_i x_i w_i^+} h}_{\text{hidden layers}} \;,\quad \underbrace{R_i = \frac{x_i + |w_i|}{\sum_i x_i + |w_i|} h}_{\text{pixel layers}}$$

Backpropagating Decompositions





Consider an arbitrary layer of a neural network, at which the neural network output $f(\boldsymbol{x})$ can be decomposed as:

$$f(\boldsymbol{x}) = \textstyle\sum_j R_j \quad \text{with} \quad R_j = h_j c_j,$$

and $c_j > 0$ *locally constant*. *Then*, $f(\boldsymbol{x})$ can also be decomposed in the previous layer:

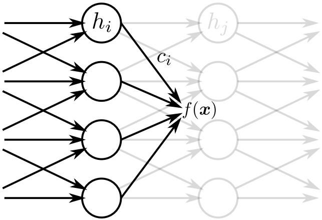
$$f(\boldsymbol{x}) = \textstyle\sum_i R_i \quad \text{with} \quad R_i = h_i c_i$$

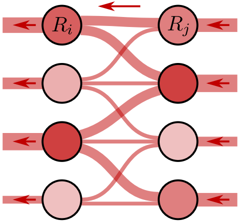
and

$$c_i = \sum_j \frac{w_{ij}^{+} h_j c_j}{\sum_i h_i w_{ij}^{+}} > 0$$

also approximately *locally constant*.

From Decomposition to Relevance Propagation





The relevance score

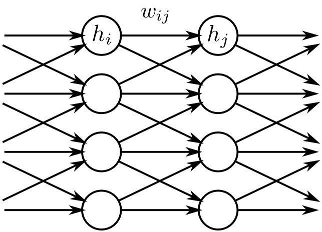
$$R_i = \underbrace{h_i \sum_j \frac{w_{ij}^+ h_j c_j}{\sum_i h_i w_{ij}^+}}_{c_i}$$

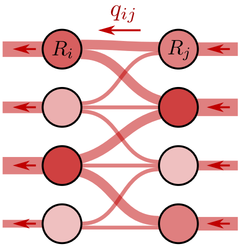
can also be written as

$$R_i = \sum_j \underbrace{\left(\frac{h_i w_{ij}^+}{\sum_i h_i w_{ij}^+}\right)}_{q_{ij}} R_j,$$

and can be interpreted as a flow of relevance propagating backwards, where q_{ij} is the fraction of relevance at unit j that flows into unit i.

Layer-Wise Relevance Propagation (LRP)





In practice, relevance propagation does *not* need to result from a *strict* deep Taylor decomposition.

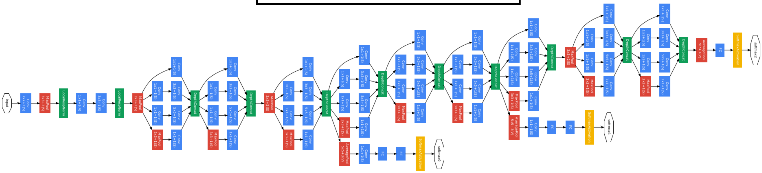
Instead, *any* propagation function $q_{ij} = g(h_i, w_{ij}, \ldots)$ with $\sum_i q_{ij} = 1$ can be used.

The propagation function can be *optimized* for some measure of *decomposition quality*.

It enables LRP's application to *various* machine learning models (e.g. Fisher-BoW + SVMs, NNs with non-ReLU units, etc.)

Layer-Wise Relevance Propagation (LRP)

step 1: forward pass
(linear time)



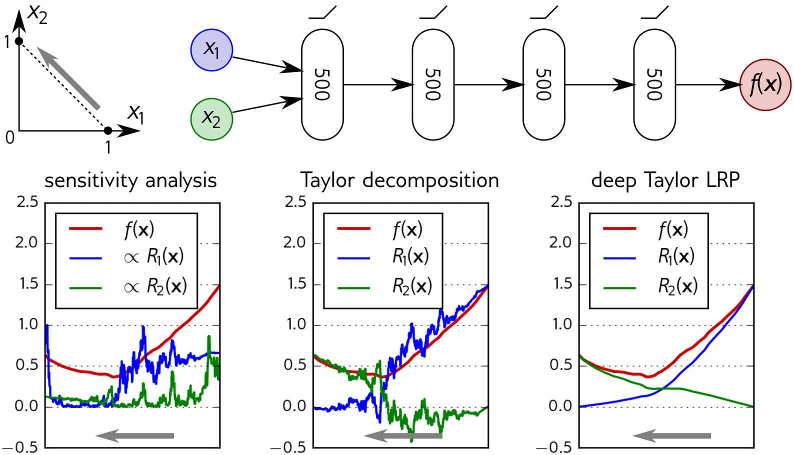
step 2: relevance propagation
also linear time!

Propagation rule:

$$R_i = \sum_j q_{ij} R_j \qquad \sum_i q_{ij} = 1$$

Various rules are available for pixel layers, intermediate layers, or special layers.

Comparing Explanation Methods



- Layer-wise relevance propagation denoises the explanation.

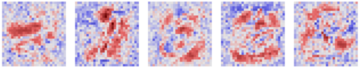
Comparison on Handwritten Digits

Data to classify:

3-layer MLP:

Sensitivity analysis

Naive Taylor ($\widetilde{\boldsymbol{x}} = 0$)

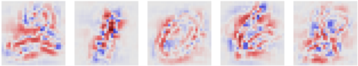


Deep Taylor LRP

6-layer CNN:

Sensitivity analysis

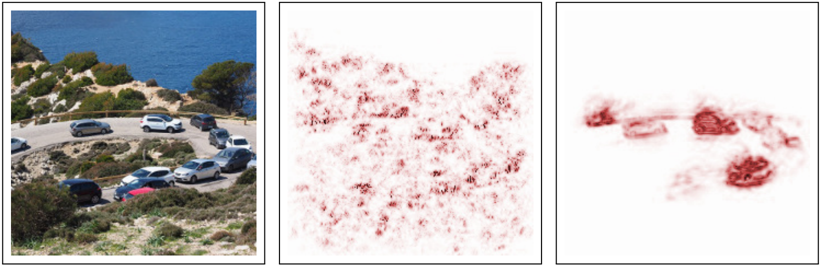
Naive Taylor ($\widetilde{\boldsymbol{x}} = 0$)



Deep Taylor LRP

Comparison on Cars Example

Image | Sensitivity Analysis | Deep Taylor LRP

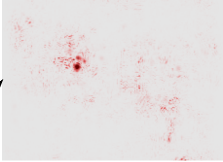


Observation: Only deep Taylor LRP focuses on cars.

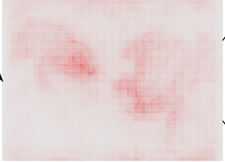
Comparison on ImageNet Models

image classified
as "frog" by
BVLC CaffeNet

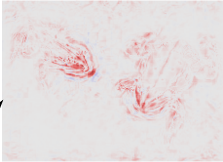
sensitivity analysis



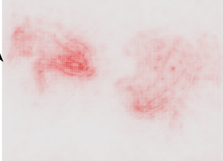
deep Taylor LRP



LRP + engineered
propagation rules ($\alpha_2\beta_1$)



deep Taylor LRP
+ better model
(GoogleNet)



Adapted from **Montavon et al. 2017**
“Explaining NonLinear Classification
Decisions with Deep Taylor Decomposition”

A Useful Trick to Implement Deep Taylor LRP

Propagation rule to implement:

$$\forall_i : \;\; R_i = \sum_j \frac{h_i w_{ij}^+}{\sum_i h_i w_{ij}^+} R_j$$

Trick: Reuse forward and backward passes from an existing implementation (e.g. Theano or TensorFlow)

$$\texttt{clone} = \texttt{layer.clone()}$$

$$\texttt{clone.W} = \max(0, \texttt{layer.W})$$

$$\texttt{clone.B} = 0$$

$$z^{(l+1)} = \texttt{clone.forward}(h^{(l)})$$

$$R^{(l)} = h^{(l)} \odot \texttt{clone.grad}(R^{(l+1)} \oslash z^{(l+1)})$$

Can be used to easily implement deep Taylor LRP in convolution and pooling layers.