ECML/PKDD 2020 Tutorial: Explainable AI for Deep Networks: Basics and Extensions

Part 3: Implementation, Theory, Evaluation, Extensions

Wojciech Samek, Grégoire Montavon

September 18, 2020

Outline of Part 3

- Implementing explanations methods
  - Automatic differentiation, backward hooks, "`.data`"
- Theoretical embedding with (deep) Taylor expansions
- Evaluating explanation methods
- Extending explanations
  - Extending beyond heatmaps
  - Extending beyond neural networks

3.a Implementation

Implementation

Implementation of different techniques can be made simple by using special techniques or tricks:

- Gradient $\times$ Input
  - Automatic differentiation
- Deconvolution
  - Backward hooks
- Layer-wise relevance propagation
  - `.detach()`

Implementing Gradient × Input

Load VGG-16 Model

In [2]:
```
import torchvision
model = torchvision.models.vgg16(pretrained=True)
model.eval();
```

Prepare to compute input gradient

In [3]:
```
X.grad = None
X.requires_grad_(True);
```

Compute explanation: $R_i = [\nabla f(\boldsymbol{x})]_i \cdot x_i$

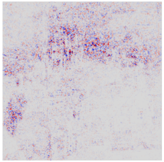
In [4]:
```
model.forward(X)[0,483].backward()
R = (X*X.grad)
```

Visualize explanation

In [5]:
```
utils.heatmap(R[0].sum(dim=0),'explanation-gi.png')
```

Input:

Output:



Implementing Deconvolution

Neuron function in a deep rectifier network:

$$z_k = \sum_{0,j} a_j w_{jk} \qquad a_k = \max(0, z_k)$$

Multivariate chain rule for derivatives (used to compute $\nabla f(\mathbf{x})$):

$$\frac{\partial f}{\partial a_j} = \sum_k \frac{\partial a_k}{\partial a_j} \frac{\partial f}{\partial a_k}$$

$$\delta_j = \sum_k w_{jk} \text{step}(z_k) \delta_k \qquad \text{(standard)}$$

Modify the backpropagation procedure:

$$\delta_j = \max(0, \sum_k w_{jk} \text{step}(z_k) \delta_k) \qquad \text{(deconvolution [14])}$$

$$\delta_j = \max(0, \sum_k w_{jk} \text{step}(z_k) \delta_k) \qquad \text{(deconvolution, guided version [12])}$$

Implementing Deconvolution (guided version, $\times$ input)

Build a hook that rectifies the gradient

```
In [6]: def hook(mod, grad_in, grad_out):
            return (grad_in[0].clamp(min=0),)
```

Register this hook in ReLU layers

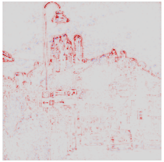
```
In [7]: for i in [1,3,6,8,11,13,15,18,20,22,25,27,29]:
            model.features[i].register_backward_hook(hook)
```

Apply Gradient $\times$ Input

```
In [8]: X.grad = None
        X.requires_grad_(True);
        model.forward(X)[0,483].backward()
        R = (X*X.grad)
        utils.heatmap(R[0].sum(dim=0),'explanation-gb.png')
```

Input:

Output:



Implementing LRP

Observation: Writing relevance scores as R_j as $a_j c_j$ and $R_k = a_k c_k$, the LRP-γ propagation rule can also be expressed as:

$$c_j = \sum_k (w_{jk} + \gamma w_{jk}^+) \frac{a_k}{p_k} c_k \qquad \text{with} \quad p_k = \textstyle\sum_{0,j} a_j (w_{jk} + \gamma w_{jk}^+)$$

and this can be further simplified to

$$c_j = \sum_k \frac{\partial p_k}{\partial a_j} \frac{a_k}{p_k} c_k = \sum_k \frac{\partial}{\partial a_j} \Big(p_k \cdot \Big[\frac{a_k}{p_k} \Big]_{\text{cst.}} \Big) c_k$$

which has the structure of the multivariate chain rule for gradient propagation.

Now, we can replace a_k by $p_k \cdot [a_k / p_k]_{\text{cst.}}$ in the forward pass and then run standard automatic differentiation get the LRP explanation [10].

Implementing LRP (simplified)

Build an equivalent forward pass where part of it is detached

In [11]:
```
class Conv(torch.nn.Module):

    def __init__(self, conv, gamma):
        torch.nn.Module.__init__(self)
        self.conv = conv
        self.pconv = copy.deepcopy(conv)
        self.pconv.weight = torch.nn.Parameter(
         conv.weight+gamma*conv.weight.clamp(min=0)
        )

    def forward(self, X):
        z = self.conv.forward(X)
        zp = self.pconv.forward(X)
        return zp * (z / zp).data
```

Implementing LRP (simplified)

Replace layers by modified layers

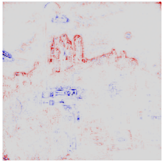
```
In [12]: f = model.features
         for i in [2]:          f[i] = Conv(f[i],1)
         for i in [5,7]:        f[i] = Conv(f[i],0.3)
         for i in [10,12,14]: f[i] = Conv(f[i],0.1)
         for i in [17,19,21]: f[i] = Conv(f[i],0.03)
         for i in [24,26,28]: f[i] = Conv(f[i],0.01)
```

Apply Gradient × Input

```
In [13]: X.grad = None
         X.requires_grad_(True);
         model.forward(X)[0,483].backward()
         R = (X*X.grad)
         utils.heatmap(R[0].sum(dim=0),'explanation-lrp.png')
```

Input:

Output:

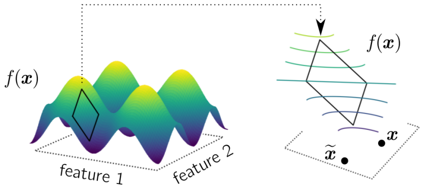


$$\widetilde{x}$$

3.b Theoretical Embedding

Taylor Expansions

- Many ML models $f(\mathbf{x})$ are complex and nonlinear when taken globally but are simple and linear when taken locally.



- The function can be approximated locally by some Taylor expansion:

$$f(\mathbf{x}) = f(\widetilde{\mathbf{x}}) + \sum_{i=1}^{d} \underbrace{[\nabla f(\widetilde{\mathbf{x}})]_i \cdot (x_i - \widetilde{x}_i)}_{R_i} + \dots$$

- First-order terms R_i of the expansion can serve as an explanation.
- The explanation $(R_i)_i$ depends on the choice of root point $\widetilde{\mathbf{x}}$.

Linear Models and Taylor Decomposition

(Homogeneous) linear model

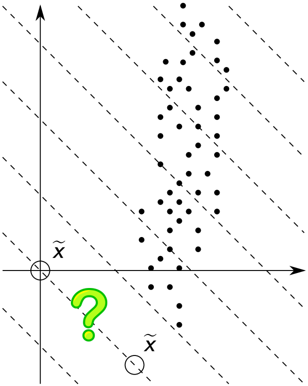
$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^\top \mathbf{x} \\ &= w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \end{aligned}$$

We first observe that for all $\mathbf{x}$:

$$\nabla f(\mathbf{x}) = \mathbf{w}$$

Then, the first-order terms of the Taylor expansion at root point $\widetilde{\mathbf{x}} = \mathbf{0}$ reduce to:

$$\begin{aligned} R_i &= [\nabla f(\widetilde{\mathbf{x}})]_i \cdot (x_i - \widetilde{x}_i) \\ &= [\mathbf{w}]_i \cdot (x_i - \widetilde{x}_i) \\ &= w_i x_i \end{aligned}$$



Gradient $\times$ Input as a Taylor Decomposition

Proposition: *When the function f is positive homogeneous, Gradient $\times$ Input corresponds to a Taylor expansion at a root point $\widetilde{\mathbf{x}} = \varepsilon \cdot \mathbf{x}$ with ε almost zero.*

Recall: we have found in Part 2 that the gradient of a positive homogeneous function is the same on any point on the segment $(\mathbf{0}, \mathbf{x})$.

Proof: We now define $\widetilde{\mathbf{x}} = \varepsilon \cdot \mathbf{x}$ a reference point with ε almost zero, we can show the connection:

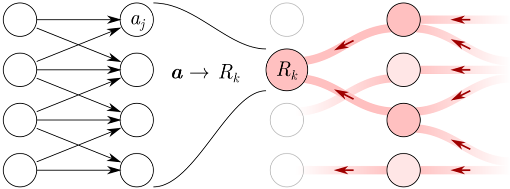
$$[\nabla f(\mathbf{x})]_i \cdot x_i \approx [\nabla f(\varepsilon \mathbf{x})]_i \cdot x_i \cdot (1 - \varepsilon) = [\nabla f(\widetilde{\mathbf{x}})]_i \cdot (x_i - \widetilde{x}_i)$$

The right hand side corresponds to the first-order terms of a Taylor expansion. $\square$

LRP as a Deep Taylor Decomposition

LRP can be embedded in the framework of deep Taylor decomposition (DTD) [7] which sees propagation as identifying linear terms of the Taylor expansion:

$$R_k(\mathbf{a}) = R_k(\widetilde{\mathbf{a}}) + \sum_j [\nabla R_k(\widetilde{\mathbf{a}})]_j \cdot (a_j - \widetilde{a}_j) + \ldots$$



Deriving the LRP-γ Rule with DTD

1. Because $R_k(\mathbf{a})$ is complicated, DTD uses the approximation:

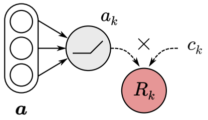
$$\widehat{R}_k(\mathbf{a}) = \big(\textstyle\sum_{0,j} a_j w_{jk}\big) \cdot c_k \qquad c_k = \text{const.}$$

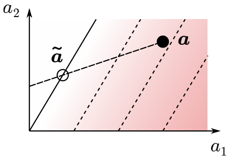
2. We choose $\widetilde{\mathbf{a}}$ on the line $\{\mathbf{a} - t\mathbf{a} \odot (\mathbf{1} + \gamma \cdot \mathbf{1}_{w_k \succeq 0}); t \in \mathbb{R}\}$. This corresponds to moving towards the origin, but faster along dimensions with positive weights.

3. Performing a Taylor expansion at $\widetilde{\mathbf{a}}$ gives the first-order terms:

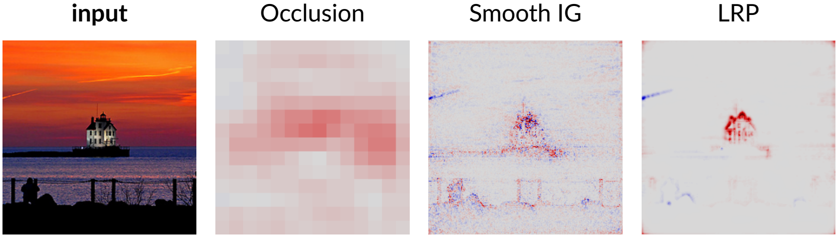
$$\begin{aligned} R_{j\leftarrow k} &= [\nabla \widehat{R}_k(\widetilde{\mathbf{a}})]_j \cdot (a_j - \widetilde{a}_j) \\ &= w_{jk} \cdot c_k \cdot t \cdot a_j \cdot (1 + \gamma \cdot 1_{w_{jk} \geq 0}) \\ &= t \cdot a_j \cdot (w_{jk} + \gamma w_{jk}^+) \cdot c_k \end{aligned}$$

4. Resolving t and applying $\sum_k$ gives the LRP-γ rule.





3.c Evaluating Explanations



Which explanation technique should be preferred?

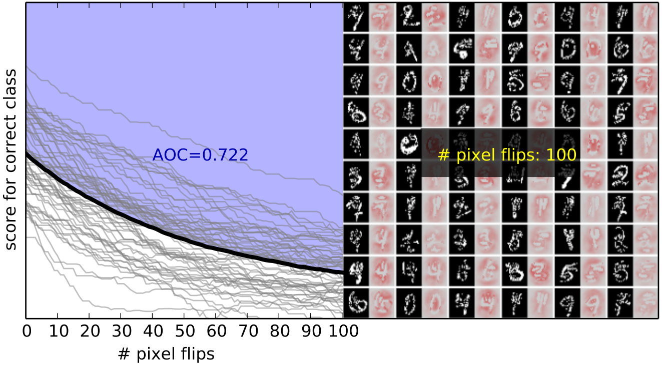
Desiderata of an Explanation

1. **Fidelity:** The explanation should reflect the quantity being explained and not something else.
2. **Understandability:** The explanation must be easily understandable by its receiver.
3. **Sufficiency:** The explanation should provide sufficient information on how the model came up with its prediction.
4. **Low Overhead:** The explanation should not cause the prediction model to become less accurate or less efficient.
5. **Runtime Efficiency:** Explanations should be computable in reasonable time.

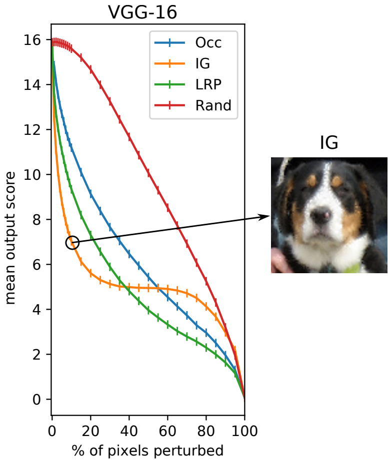
(cf. Swartout & Moore 1993 [13])

Evaluating Fidelity: Pixel-Flipping

- The pixel-flipping procedure [9] destroys pixels from most to least relevant according to the explanation, and keeps track of the neural network output.
- The faster the output decreases, the better the explanation.

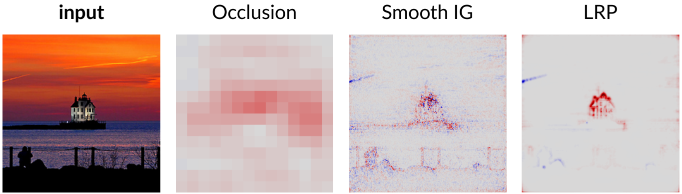


Evaluating Fidelity: Pixel-Flipping on VGG-16



- All explanation methods are more faithful than a random explanation.
- IG is the most faithful for the first few most relevant pixels, and then stagnates.
- Although not detected by VGG-16 anymore, the class-relevant patterns are still there after flipping (e.g. we can still see the dog). Did IG actually explain a *vulnerability* of VGG-16 instead of its typical behavior?

Evaluating Understandability: File Size



- A simple proxy quantity for understandability is *average file size* (the smaller, the easier to understand) [10]:

	Occ	IG	LRP
VGG-16	**698.4**	5795.0	1828.3
ResNet-50	**693.6**	5978.0	2928.2

- Better measures based on some human perceptual model, or some cognitive experiment, can be designed (e.g. [5]).

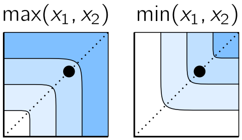
Evaluating Sufficiency

- Example of a faithful, understandable, but *insufficient* explanation

  Q: *Why did the classifier predict this image to be a 'lighthouse'?*
  A: *Because the classifier found a lighthouse in the image.*

- Evaluating sufficiency:
  - Is the explanation actionable? (e.g. can we improve a model from the produced explanations).
  - Can we learn something general about the classifier? (e.g. what kind of features it uses).

- Is it sufficient to explain a prediction in terms of individual pixels, or should we identify higher-order interactions?

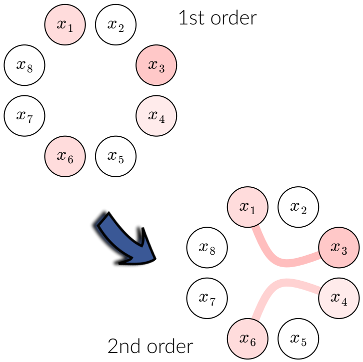


$$\nabla^2$$

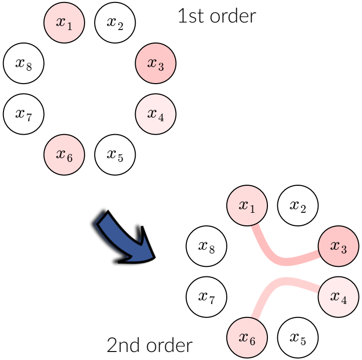
3.d Extending Explanations Beyond Heatmaps

From 1st-Order to Higher-Order Explanations

- First-order explanations support basic reasoning (input features contribute additively to the prediction).
- Many real-world predictions occur due to a conjunction of factors (e.g. two objects being present simultaneously in the data).
- These conjunctions can be captured by high-order explanations.



Explanation with 2nd-Order Taylor Expansions



2nd-order Taylor expansion

$$
\begin{aligned}
f(\mathbf{x}) = f(\widetilde{\mathbf{x}}) & \\
& + \textstyle\sum_i [\nabla f(\widetilde{\mathbf{x}})]_i \, (x_i - \widetilde{x}_i) \\
& \quad + \textstyle\sum_{ii'} \frac{1}{2} [\nabla^2 f(\widetilde{\mathbf{x}})]_{ii'} \, (x_i - \widetilde{x}_i)\,(x_{i'} - \widetilde{x}_{i'}) \\
& \qquad + \ldots
\end{aligned}
$$

2nd-order *deep* Taylor expansion

$$
\begin{aligned}
R_{kk'}(\mathbf{a}) = R_{kk'}(\widetilde{\mathbf{a}}) & \\
& + \textstyle\sum_j [\nabla R_{kk'}(\widetilde{\mathbf{a}})]_j \cdot (a_j - \widetilde{a}_j) \\
& \quad + \textstyle\sum_{jj'} \frac{1}{2} [\nabla^2 R_{kk'}(\widetilde{\mathbf{a}})]_{jj'} \cdot (a_j - \widetilde{a}_j)\,(a_{j'} - \widetilde{a}_{j'}) \\
& \qquad + \ldots
\end{aligned}
$$

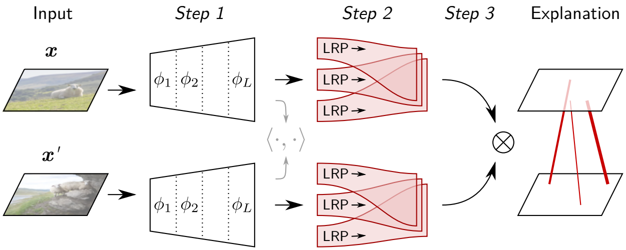
Explaining Similarity with BiLRP [1]

- Applies to dot-product similarities of the type

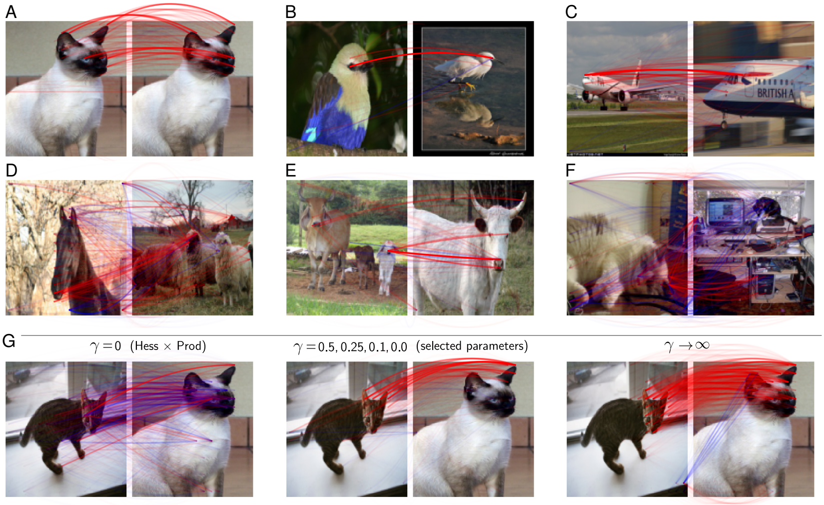
$$y(\mathbf{x}, \mathbf{x}') = \langle \phi_L \circ \cdots \circ \phi_1(\mathbf{x}), \phi_L \circ \cdots \circ \phi_1(\mathbf{x}') \rangle$$

where $\phi_L \circ \cdots \circ \phi_1$ is a deep rectifier network.

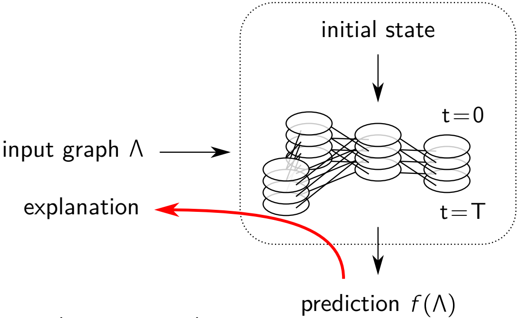
- Performs a 2nd-order (deep) Taylor decomposition of the similarity score. The procedure factorizes into an composition of multiple standard LRP computations.



Explaining Similarity with BiLRP [1]



Explaining Graph Neural Networks



High-order Taylor expansion to decompose the prediction in terms of ‘relevant walks’ [11]:

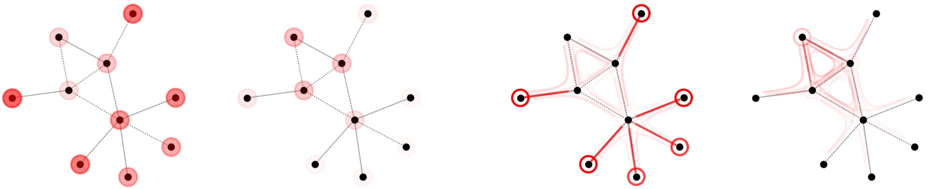
$$R_{\mathcal{W}} = \frac{\partial^{|\mathcal{W}|} f}{\ldots \partial \lambda_{JK} \ldots}\bigg|_{\boldsymbol{\Lambda}=\widetilde{\boldsymbol{\Lambda}}} \cdot \left[\ldots \cdot (\lambda_{JK} - \widetilde{\lambda}_{JK}) \cdot \ldots\right]$$

Explaining Graph Neural Networks

Example:

- Explaining why an input graph **x** is predicted by some GNN to be a Barabási-Albert (BA) graph of growth parameter 1 or 2 (i.e. “tree” or “not tree”).

1st-order explanation [8]		High-order explanation (GNN-LRP) [11]	
evidence for "tree"	evidence for "not tree"	evidence for "tree"	evidence for "not tree"



$$k(\cdot, \cdot)$$

3.e Extending Explanations Beyond Neural Nets

Extending Explanations Beyond Neural Networks

Observation:

- Non-neural network algorithms such as kernel machines remain popular for unsupervised tasks, e.g. kernel density estimation, one-class SVMs, kernel k-means.

Two possible approaches:

1. Adapt explanation methods to handle these kernel models.
2. Rewrite these models as neural networks [2, 3, 4] ('**neuralize**' them).

Neuralizing Kernel Density Models [3, 4]

Kernel density estimation (KDE) and one-class SVMs are non-neural network models for density estimation / anomaly detection. The inlier score can be generically written as a weighted sum of kernel scores:

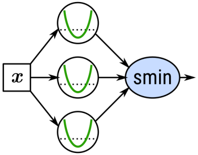
$$f(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j \exp(-\gamma \|\mathbf{x} - \mathbf{x}_j\|^2)$$

If interested in detecting *anomaly*, we can consider instead the quantity $o(\mathbf{x}) = -\log f(\mathbf{x})$.

This quantity can be rewritten as a strictly equivalent two-layer neural network:

$$h_j = \gamma \|\mathbf{x} - \mathbf{x}_j\|^2 - \log \alpha_j \qquad \text{(layer 1)}$$

$$o(\mathbf{x}) = \underbrace{-\log\Big(\sum_{j=1}^{N} \exp(-}_{\text{smin}} h_j)\Big) \qquad \text{(layer 2)}$$



Standard explanation techniques for neural networks (e.g. LRP) can now be applied.

Neuralizing Log-Likelihood Ratios [2, 6]

Class or cluster membership probabilities are often modeled via the 'softmax' function:

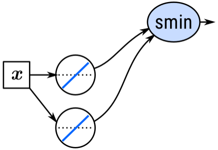
$$p_k = \frac{\exp(\mathbf{w}_k^\top \mathbf{a})}{\sum_j \exp(\mathbf{w}_j^\top \mathbf{a})}$$

Because softmax saturates at 0 and 1, it doesn't capture the full evidence for/against the class. The log-likelihood ratio $\ell_k = \log(p_k/(1-p_k))$ does not saturate.

This quantity can be rewritten as a strictly equivalent two-layer neural network:

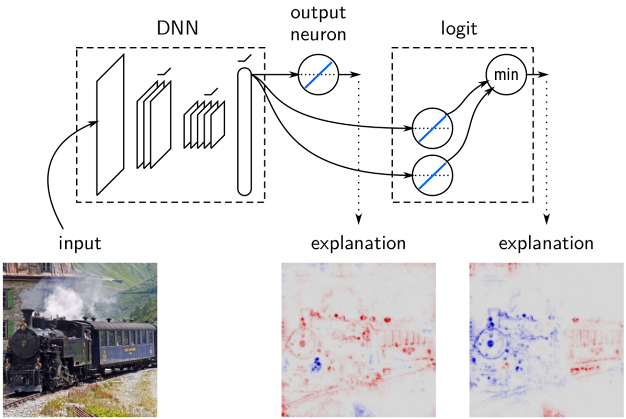
$$h_j = (\mathbf{w}_k - \mathbf{w}_j)^\top \mathbf{a} \qquad \text{(layer 1)}$$

$$\ell_k(\mathbf{a}) = \underbrace{-\log \textstyle\sum_{j \neq k} \exp(-h_j)}_{\text{smin}} \qquad \text{(layer 2)}$$



Again, explanation techniques for neural networks (e.g. LRP) can now be applied.

Example: Explaining 'Passenger Car'



- We explain the output before and after the log-likelihood ratio (logit).
- Locomotive is correlated to the passenger_car, but it *lowers* the probability of the class passenger_car, because it raises the probability of the class locomotive.

Neuralizing Kernel K-Means [2]

Kernel k-means model (KDE + softmax)

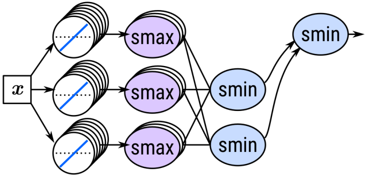
$$p_c = \frac{\left(Z_c^{-1} \sum_{i \in \mathcal{C}_c} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2)\right)^{\beta/\gamma}}{\sum_k \left(Z_k^{-1} \sum_{j \in \mathcal{C}_k} \exp(-\gamma \|\mathbf{x} - \mathbf{x}_j\|^2)\right)^{\beta/\gamma}}$$

Again, this model can be rewritten as a strictly equivalent neural network composed of a linear layer and a succession of pooling layers.

$$\log\left[\frac{p_c}{1 - p_c}\right] = \beta \, \underset{k \neq c}{\mathrm{smin}}^{\beta}\Big\{\underset{j \in \mathcal{C}_k}{\mathrm{smin}}^{\gamma}\Big\{\underset{i \in \mathcal{C}_c}{\mathrm{smax}}^{\gamma}\big\{\mathbf{w}_{ij}^\top \mathbf{x} + b_{ijk}\big\}\Big\}\Big\}$$

with

- $\mathbf{w}_{ij} = 2(\mathbf{x}_i - \mathbf{x}_j)$
- $b_{ijk} = \|\mathbf{x}_j\|^2 - \|\mathbf{x}_i\|^2 + \gamma^{-1}(\log Z_k - \log Z_c)$
- $\mathrm{smin}_j^{\gamma}\{\cdot\} = -\gamma^{-1} \log \sum_j \exp(-\gamma(\cdot))$



Summary

- Explanation methods are easy to *implement* when using the proper tricks (backward hooks, `.detach()`).
- Explanation methods can be cast into the *theoretical* framework of Taylor expansions.
- *Evaluating* explanations requires to test multiple factors (fidelity, understandability, sufficiency, ...)
- When heatmaps are not sufficient, explanations can be *extended* using higher-order Taylor expansions.
- Some models that are not neural networks (e.g. kernel-based) can be converted into a strictly equivalent neural networks (or *'neuralized'*), so that explanation techniques such as LRP can be applied.

References I

[1] O. Eberle, J. Büttner, F. Kräutli, K.-R. Müller, M. Valleriani, and G. Montavon.
Building and interpreting deep similarity models.
CoRR, abs/2003.05431, 2020.

[2] J. Kauffmann, M. Esders, G. Montavon, W. Samek, and K.-R. Müller.
From clustering to cluster explanations via neural networks.
CoRR, abs/1906.07633, 2019.

[3] J. Kauffmann, K.-R. Müller, and G. Montavon.
Towards explaining anomalies: A deep taylor decomposition of one-class models.
Pattern Recognit., 101:107198, 2020.

[4] J. Kauffmann, L. Ruff, G. Montavon, and K.-R. Müller.
The clever hans effect in anomaly detection.
CoRR, abs/2006.10609, 2020.

[5] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. Gershman, and F. Doshi-Velez.
An evaluation of the human-interpretability of explanation.
CoRR, abs/1902.00006, 2019.

References II

[6] 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.

[7] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller.
Explaining nonlinear classification decisions with deep taylor decomposition.
Pattern Recognit., 65:211–222, 2017.

[8] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann.
Explainability methods for graph convolutional neural networks.
In *CVPR*, pages 10772–10781. Computer Vision Foundation / IEEE, 2019.

[9] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K. Müller.
Evaluating the visualization of what a deep neural network has learned.
IEEE Trans. Neural Networks Learn. Syst., 28(11):2660–2673, 2017.

[10] 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.

References III

[11] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K.-R. Müller, and G. Montavon.
XAI for graphs: Explaining graph neural network predictions by identifying relevant walks.
CoRR, abs/2006.03589, 2020.

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

[13] W. R. Swartout and J. D. Moore.
Explanation in Second Generation Expert Systems, page 543585.
Springer-Verlag, Berlin, Heidelberg, 1993.

[14] 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.