Part 3: Implementation, Theory, Evaluation, Extensions

Wojciech Samek, Grégoire Montavon

September 18, 2020
Outline of Part 3

- Implementing explanations methods
  - Automatic differentiation, backward hooks, "\texttt{.data}"
- Theoretical embedding with (deep) Taylor expansions
- Evaluating explanation methods
- Extending explanations
  - Extending beyond heatmaps
  - Extending beyond neural networks
3.a 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 $\times$ Input

Load VGG-16 Model

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

Prepare to compute input gradient

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

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

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

Visualize explanation

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

Neuron function in a deep rectifier network:

\[ z_k = \sum_{0,j} a_j w_{jk} \quad a_k = \max(0, z_k) \]

Multivariate chain rule for derivatives (used to compute \( \nabla f(x) \)):

\[
\frac{\partial f}{\partial a_j} = \sum_k \frac{\partial a_k}{\partial a_j} \frac{\partial f}{\partial a_k} \\
\delta_j = \sum_k w_{jk} \text{step}(z_k) \delta_k \quad (\text{standard})
\]

Modify the backpropagation procedure:

\[
\delta_j = \max(0, \sum_k w_{jk} \text{step}(z_k) \delta_k) \quad (\text{deconvolution [14]}) \\
\delta_j = \max(0, \sum_k w_{jk} \text{step}(z_k) \delta_k) \quad (\text{deconvolution, guided version [12]})
\]
Implementing Deconvolution (guided version, × 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 × 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')
Implementing LRP

**Observation:** Writing relevance scores as $R_j$ as $a_jc_j$ and $R_k = a_kc_k$, the LRP-$\gamma$ propagation rule can also be expressed as:

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

and this can be further simplified to

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

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

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

Apply Gradient \times Input

\[
\text{In [13]: } \quad X.\text{grad} = \text{None} \\
X.\text{requires_grad}(_\text{True}); \\
\text{model.forward}(X)[0,483].\text{backward}() \\
R = (X*X.\text{grad}) \\
\text{utils.heatmap}(R[0].\text{sum(dim}=0),_\text{explanation-lrp.png})
\]
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:

\[
    f(\mathbf{x}) = f(\tilde{\mathbf{x}}) + \sum_{i=1}^{d} [\nabla f(\tilde{\mathbf{x}})]_i \cdot (x_i - \tilde{x}_i) + \ldots
\]

- 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 $\tilde{\mathbf{x}}$. 
(Homogeneous) linear model

\[ f(x) = \mathbf{w}^\top x \]
\[ = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \]

We first observe that for all \( x \):

\[ \nabla f(x) = \mathbf{w} \]

Then, the first-order terms of the Taylor expansion at root point \( \tilde{x} = 0 \) reduce to:

\[ R_i = [\nabla f(\tilde{x})]_i \cdot (x_i - \tilde{x}_i) \]
\[ = [\mathbf{w}]_i \cdot (x_i - \tilde{x}_i) \]
\[ = w_i x_i \]
**Proposition:** When the function $f$ is positive homogeneous, Gradient $\times$ Input corresponds to a Taylor expansion at a root point $\tilde{x} = \epsilon \cdot x$ with $\epsilon$ 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 $(0, x)$.

**Proof:** We now define $\tilde{x} = \epsilon \cdot x$ a reference point with $\epsilon$ almost zero, we can show the connection:

$$[\nabla f(x)]_i \cdot x_i \approx [\nabla f(\epsilon x)]_i \cdot x_i \cdot (1 - \epsilon) = [\nabla f(\tilde{x})]_i \cdot (x_i - \tilde{x}_i)$$

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

$$R_k(a) = R_k(\tilde{a}) + \sum_j [\nabla R_k(\tilde{a})]_j \cdot (a_j - \tilde{a}_j) + \ldots$$
1. Because $R_k(a)$ is complicated, DTD uses the approximation:

$$\hat{R}_k(a) = (\sum_{0,j} a_j w_{jk}) \cdot c_k \quad c_k = \text{const.}$$

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

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

$$R_{j \leftarrow k} = [\nabla \hat{R}_k(\tilde{a})]_j \cdot (a_j - \tilde{a}_j)$$

$$= w_{jk} \cdot c_k \cdot t \cdot a_j \cdot (1 + \gamma \cdot 1_{w_j \geq 0})$$

$$= t \cdot a_j \cdot (w_{jk} + \gamma w_{jk}^+) \cdot c_k$$

4. Resolving $t$ and applying $\sum_k$ gives the LRP-$\gamma$ 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.
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?
A simple proxy quantity for understandability is *average file size* (the smaller, the easier to understand) [10]:

<table>
<thead>
<tr>
<th></th>
<th>Occ</th>
<th>IG</th>
<th>LRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>698.4</td>
<td>5795.0</td>
<td>1828.3</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>693.6</td>
<td>5978.0</td>
<td>2928.2</td>
</tr>
</tbody>
</table>

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?
3.d Extending Explanations
Beyond Heatmaps
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

\[
    f(x) = f(\bar{x}) + \sum_i \left[ \nabla f(\bar{x}) \right]_i (x_i - \bar{x}_i) + \sum_{ii'} \frac{1}{2} \left[ \nabla^2 f(\bar{x}) \right]_{ii'} (x_i - \bar{x}_i)(x_{i'} - \bar{x}_{i'}) + \ldots
\]

2nd-order deep Taylor expansion

\[
    R_{kk'}(a) = R_{kk'}(\bar{a}) + \sum_j \left[ \nabla R_{kk'}(\bar{a}) \right]_j \cdot (a_j - \bar{a}_j) + \sum_{jj'} \frac{1}{2} \left[ \nabla^2 R_{kk'}(\bar{a}) \right]_{jj'} \cdot (a_j - \bar{a}_j)(a_{j'} - \bar{a}_{j'}) + \ldots
\]
Explaining Similarity with BiLRP [1]

- Applies to dot-product similarities of the type

\[ y(x, x') = \langle \phi_L \circ \cdots \circ \phi_1(x), \phi_L \circ \cdots \circ \phi_1(x') \rangle \]

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 a composition of multiple standard LRP computations.

![Diagram](image-url)
Explaining Similarity with BiLRP [1]
High-order Taylor expansion to decompose the prediction in terms of ‘relevant walks’ [11]:

\[ R_W = \frac{\partial^{|W|} f}{\partial \lambda_{JK} \cdots} \bigg|_{\Lambda = \tilde{\Lambda}} \cdot \left[ \cdots \left( \lambda_{JK} - \tilde{\lambda}_{JK} \right) \cdots \right] \]
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”).

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>evidence for &quot;tree&quot;</td>
<td>evidence for &quot;not tree&quot;</td>
</tr>
<tr>
<td>evidence for &quot;not tree&quot;</td>
<td>evidence for &quot;not tree&quot;</td>
</tr>
</tbody>
</table>

Example:
3.e Extending Explanations Beyond Neural Nets
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:

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

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

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

$$h_j = \gamma \|x - x_j\|^2 - \log \alpha_j$$  \hspace{1cm} \text{(layer 1)}

$$o(x) = -\log \left( \sum_{j=1}^{N} \exp(-h_j) \right)$$  \hspace{1cm} \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:

\[ p_k = \frac{\exp(w_k^\top a)}{\sum_j \exp(w_j^\top a)} \]

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:

\[ h_j = (w_k - w_j)^\top a \]  \hspace{1cm} (layer 1)
\[ \ell_k(a) = -\log \sum_{j \neq k} \exp(-h_j) \]  \hspace{1cm} (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.

Kernel k-means model (KDE + softmax)

\[ p_c = \frac{(Z_c^{-1} \sum_{i \in C_c} \exp(-\gamma \|x - x_i\|^2))^{\beta/\gamma}}{\sum_k (Z_k^{-1} \sum_{j \in C_k} \exp(-\gamma \|x - x_j\|^2))^{\beta/\gamma}} \]

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

\[ \log\left(\frac{p_c}{1 - p_c}\right) = \beta \min_{k \neq c} \{s_{min}^\gamma \{\min_{j \in C_k} \{s_{max}^\gamma \{w_{ij}^\top x + b_{ijk}\}\}\}\} \]

with

- \( w_{ij} = 2(x_i - x_j) \)
- \( b_{ijk} = \|x_j\|^2 - \|x_i\|^2 + \gamma^{-1}(\log Z_k - \log Z_c) \)
- \( s_{min}^\gamma \{\cdot\} = -\gamma^{-1} \log \sum_j \exp(-\gamma(\cdot)) \)
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

Building and interpreting deep similarity models.

From clustering to cluster explanations via neural networks.

Towards explaining anomalies: A deep taylor decomposition of one-class models.

The clever hans effect in anomaly detection.

An evaluation of the human-interpretability of explanation.
Layer-wise relevance propagation: An overview.

Explaining nonlinear classification decisions with deep taylor decomposition.


Evaluating the visualization of what a deep neural network has learned.

Toward interpretable machine learning: Transparent deep neural networks and beyond.
XAI for graphs: Explaining graph neural network predictions by identifying relevant walks. 

Striving for simplicity: The all convolutional net. 
In ICLR (Workshop), 2015.


Visualizing and understanding convolutional networks. 