

# Model Interpretability

---

## Model Interpretability

[Todo List](#)

[On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation](#)

[Contribution](#)

[Links](#)

[Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers](#)

[Contribution](#)

[Notes](#)

[Links](#)

[SmoothGrad: removing noise by adding noise](#)

[Contributes](#)

[Notes](#)

[Links](#)

[Axiomatic Attribution for Deep Networks](#)

[Contribution](#)

[Notes](#)

[Links](#)

[Visualizing and Understanding Neural Machine Translation](#)

[Contribution](#)

[Notes](#)

[Links](#)

[SHAP - SHapley Additive exPlanation](#)

[Contribution](#)

[Notes](#)

[SHAP 的基本运作原理](#)

[Game Theory and Machine Learning](#)

[A Power Set of Features](#)

[Marginal Contributions of a Feature](#)

[Weighting the Marginal Contributions](#)

[Wrapping it up](#)

[Kernel SHAP \(Linear LIME + Shapley Values\)](#)

[Gradient SHAP](#)

[Tree SHAP](#)

[Tree Prediction on Feature Missing Instance](#)

[Links](#)

[Beyond Sparsity: Tree Regularization of Deep Models for Interpretability](#)

[Contribution](#)

[Links](#)

[LEMNA: Explaining Deep Learning based Security Applications](#)

[Contribution](#)

[Notes](#)

[Links](#)

[Robust Classification with Convolutional Prototype Learning](#)

[Contribution](#)

[Links](#)

[Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its](#)

[Predictions](#)

[Contribution](#)

[Notes](#)

[Links](#)

[机器学习模型可解释性方法、应用与安全研究综述](#)

[Notes](#)

机器学习可解释性问题

\* Ante-hoc 可解释性

Post-hoc 可解释性

可解释性应用

可解释性与安全性分析

未来方向

Links

[Regional Tree Regularization for Interpretability in Black Box Models](#)

Contribution

Notes

Links

[Layer-Wise Relevance Propagation: An Overview](#)

Notes

Links

## Todo List

---

1. Bach S, Binder A, Montavon G, et al. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation [J]. *PloS one*, 2015, 10(7): e0130140.
2. Guo, Wenbo, et al. "Lemna: Explaining deep learning based security applications." *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018.
3. Tao Guanhong, Ma Shiqing, Liu Yingqi, et al. Attacks meet interpretability: Attribute-steered detection of adversarial samples [C] //Proc of the 32st Int Conf on Neural Information Processing Systems. USA: Curran Associates Inc., 2018: 7717-7728
4. Liu Ninghao, Yang Hongxia, Hu Xia. Adversarial detection with model interpretation [C] //Proc of the 24th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2018: 1803-1811
5. Carlini N, Wagner D. Towards evaluating the robustness of neural networks [C] //Proc of the 38th IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2017: 39-57
6. Papernot N, McDaniel P, Jha S, et al. The limitations of deep learning in adversarial settings [C] //Proc of the 1st IEEE European Symp on Security and Privacy. Piscataway, NJ: IEEE, 2016: 372-387
7. Papernot N, McDaniel P, Goodfellow I, et al. Practical blackbox attacks against machine learning [C] //Proc of the 12th ACM Asia Conf on Computer and Communications Security. New York: ACM, 2017: 506-519
8. Ghorbani A, Abid A, Zou J. Interpretation of neural networks is fragile [J]. arXiv preprint arXiv:1710.10547, 2017
9. Zhang Xinyang, Wang Ningfei, Ji Shouling, et al. Interpretable Deep Learning under Fire [C] //Proc of the 29th USENIX Security Symp. Berkele, CA: USENIX Association, 2020
10. GRADIENTS OF COUNTERFACTUALS(ICLR 2017)
11. Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034, 2013.
12. Explainable Neural Network based on Generalized Additive Model.
13. Montavon, G., Lapuschkin, S., Binder, A., Samek, W., M\"uller, K.R.: Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recogn.* 65, 211-222 (2017)

# On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation

## Contribution

- 作者提出了 LRP 的可解释性方法;

## Links

- 论文链接: [Bach S, Binder A, Montavon G, et al. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation\[J\]. PloS one, 2015, 10\(7\): e0130140.](#)

## Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers

## Contribution

- 作者将 LRP 可解释方法扩展到了 Local Renormalization Layer 这种非线性的网络层;

## Notes

1. LRP (Layer-wise Relevance Propagation) : 该方法利用 **神经元之间的数值关系**, 回传 **相关性系数** (注意和梯度是有区别的, 我认为他和梯度的方法最大的不同点在于, 该方法考虑了神经元之间的相关性)

(1) 神经元的激活公式如下:

$$x_j = g\left(\sum_i w_{ij} x_i + b\right)$$

(2) 给定一张图片  $x$  和一个神经网络  $f$ , LRP 方法的目的在于给图像的每一个像素点都给定一个 **相关性系数**:

$$f(x) \approx \sum_p R_p^{(1)}$$

(3) 给定神经网络  $l+1$  层的神经元, 它的 **相关性系数** 存在如下公式:

$$R_j^{(l+1)} = \sum_{i \in (l)} R_{i \leftarrow j}^{(l, l+1)}$$

(4) 那么, 我们就可以得到  $l$  层的神经元, 它的 **相关性系数** 存在如下公式: (即完成了系数的回传)

$$R_i^{(l)} = \sum_{j \in (l+1)} R_{i \leftarrow j}^{(l, l+1)}$$

其中, 最后一层的相关性系数等于  $f(x)$ ;

(5) 在 [前文](#) 中, 有两套公式来计算系数的回传:

- $\epsilon - rule$ :

$$R_{i \leftarrow j}^{(l, l+1)} = \frac{z_{ij}}{z_j + \epsilon \cdot \text{sign}(z_j)} R_j^{(l+1)}$$

其中,  $z_{ij} = (w_{ij} x_i)^p$  and  $z_j = \sum_{k: w_{kj} \neq 0} z_{kj}$ ;  $\epsilon$  用来防止除 0 ;

- $\beta$  – rule:

$$R_{i \leftarrow j}^{(l,l+1)} = \left( (1 + \beta) \frac{z_{ij}^+}{z_j^+} - \beta \frac{z_{ij}^-}{z_j^-} \right) R_j^{(l+1)}$$

其中,  $\beta$  用来控制相关性再分配中对负相关系数的关注程度, 其值越大生成的热力图越锐化;

- 合并上面两套公式:

$$R_{i \leftarrow j}^{(l,l+1)} = v_{ij} R_j^{(l+1)} \quad \text{with} \quad \sum_i v_{ij} = 1$$

2. Extending LRP: 将 LRP 方法扩展到 LRN 层;

(1) 扩展的原因: 我们先看一下 LRN 层的激活公式

$$y_k(x_1, \dots, x_n) = \frac{x_k}{(1 + b \sum_{i=1}^n x_i^2)^c}$$

很显然, 它不满足第一部分讲得一般的神经网络激活公式;

(2) ☆ 泰勒展开式: 假设存在  $x_j = g(x_{h_1}, \dots, x_{h_n})$ , 我们可以对其使用一阶泰特展开式展开

$$x_j \approx g(\tilde{x}_{h_1}, \dots, \tilde{x}_{h_n}) + \sum_{i \leftarrow j} \frac{\partial g}{\partial x_{h_i}}(\tilde{x}_{h_1}, \dots, \tilde{x}_{h_n})(x_{h_i} - \tilde{x}_{h_i})$$

那么, 我们可以得到相关性系数的公式为

$$\forall_{i \leftarrow j}: z_{ij} = \frac{1}{n} g(\tilde{x}_{h_1}, \dots, \tilde{x}_{h_n}) + \frac{\partial g}{\partial x_{h_i}}(\tilde{x}_{h_1}, \dots, \tilde{x}_{h_n})(x_{h_i} - \tilde{x}_{h_i})$$

(3) 推导 LRN 层: 对  $y_k$  求偏导,

$$\frac{\partial y_k}{\partial x_j} = \begin{cases} \frac{1}{(1+b \sum_{i=1}^n x_i^2)^c} - \frac{2bcx_j x_k}{(1+b \sum_{i=1}^n x_i^2)^{c+1}}, & j = k \\ -\frac{2bcx_j x_k}{(1+b \sum_{i=1}^n x_i^2)^{c+1}}, & j \neq k \end{cases}$$

整理一下就可以得到原文公式

$$\frac{\partial y_k}{\partial x_j} = \frac{\delta_{kj}}{(1 + b \sum_{i=1}^n x_i^2)^c} - 2bc \frac{x_k x_j}{(1 + b \sum_{i=1}^n x_i^2)^{c+1}}$$

(4) 特殊泰勒展开点: (我很困惑, 为什么明知道这个展开点不好展开, 还要在这个点去进行展开) 假设现在有实际输入  $z_1 = (x_1, x_2, \dots, x_n)$  和泰勒展开点  $z_2 = (0, 0, \dots, x_k, \dots, 0)$ , 不难得到如下公式

$$y_k(z_1) \approx y_k(z_2) + 0 = \frac{x_k}{(1 + bx_k^2)^c}$$

可以看到, 这里的泰特一阶展开式是无效的。所以, 下面对其进行相应的修改

$$\begin{aligned} y_k(z_2) &\approx y_k(z_1) + \nabla y_k(z_1) \cdot (z_2 - z_1) \\ \Rightarrow y_k(z_1) &\approx y_k(z_2) + \nabla y_k(z_1) \cdot (z_1 - z_2) \\ \Rightarrow y_k(z_1) &\approx \frac{x_k}{(1 + bx_k^2)^c} - 2bc \sum_{j:j \neq k} \frac{x_k x_j^2}{(1 + b \sum_{i=1}^n x_i^2)^{c+1}} \end{aligned}$$

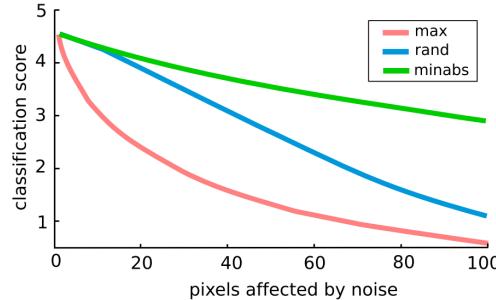
然后可以用 (2) 进行回传;

3. 实验: (只简单地看一下其中两个实现结果)

(1) 验证得到的重要特征是否是有效的

- 检验手段: 模糊化目标像素点, 观察模型置信度的变化;

- 实验结果：文章的方法可以找到图像中的重要特征点；



**Fig. 2.** Decrease of classification score as pixels are sequentially replaced by random noise on the CIFAR-10 dataset. Red curve: pixels with highest pixel-wise scores are flipped first. Blue curve: pixels are flipped in random order. Green curve: least relevant pixels are flipped first. A similar comparison for Imagenet is found in [8].

- (2) 检验文章方法是否优于原有方法（原有方法直接将相关性权重整个回传给中心特征点  $x_k$ ）

- 检验手段：同样模糊化重要特征，观察模型 AUC 的变化（越小越好）；
- 实验结果：文章方法更优；

| rule for basic layers      | rule for normalization layers | AUC score |
|----------------------------|-------------------------------|-----------|
| eq. 4,5, $\epsilon = 0.01$ | identity                      | 37.10     |
| eq. 4,5, $\epsilon = 0.01$ | first-order Taylor            | 35.47     |
| eq. 4,6, $\beta = 1$       | identity                      | 56.13     |
| eq. 4,6, $\beta = 1$       | first-order Taylor            | 53.82     |

**Table 1.** Comparison of different types of LRN layer treatments for two approaches of computing pixel-wise scores for CIFAR-10. Lower scores are better.

## Links

- 论文链接：[Binder A, Montavon G, Lapuschkin S, et al. Layer-wise relevance propagation for neural networks with local renormalization layers\[C\]//International Conference on Artificial Neural Networks. Springer, Cham, 2016: 63-71.](#)
- 参考链接：[Tensorflow的LRN是怎么做的](#)
- 参考链接：[【阅读笔记】神经网络中的LRP及其在非线性神经网络中的运用](#)
- 参考链接：[Implementation of LRP for pytorch](#)

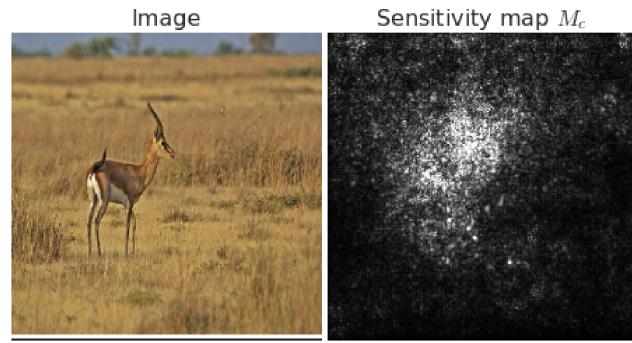
## SmoothGrad: removing noise by adding noise

### Contributes

1. 尝试解决梯度可解释性方法中存在的 **梯度饱和问题**，在一定程度上去除 Grad 方法中的噪声；

### Notes

1. 已有的基于梯度的算法能够得到一个重要性关联的图，但是带有很多的噪声，即直接计算梯度，会出现 **梯度饱和的问题** (在当前图中，羊的特征对当前的分类概率影响不大，因为它的分类结果可能已经是0.999了)



2. 作者提出的想法是添加扰动后再对图片求梯度, 然后将这些梯度求一个均值, 故论文名称为  
Removing noise by adding noise, 求导的时候添加扰动, 是为了除去在重要性结果上的噪声:

$$\hat{M}_c(x) = \frac{1}{n} \sum_1^n M_c(x + \mathcal{N}(0, \sigma^2))$$

### 3. Evaluation:

- (1) 模型+数据集: Inception v3 trained on ILSVRC-2013 dataset, convolutional MNIST model  
(2) 噪声大小的影响: 噪声过大的情况下也会使得效果变差, 从图中的效果来看, 添加 10% 的噪声效果是最好的

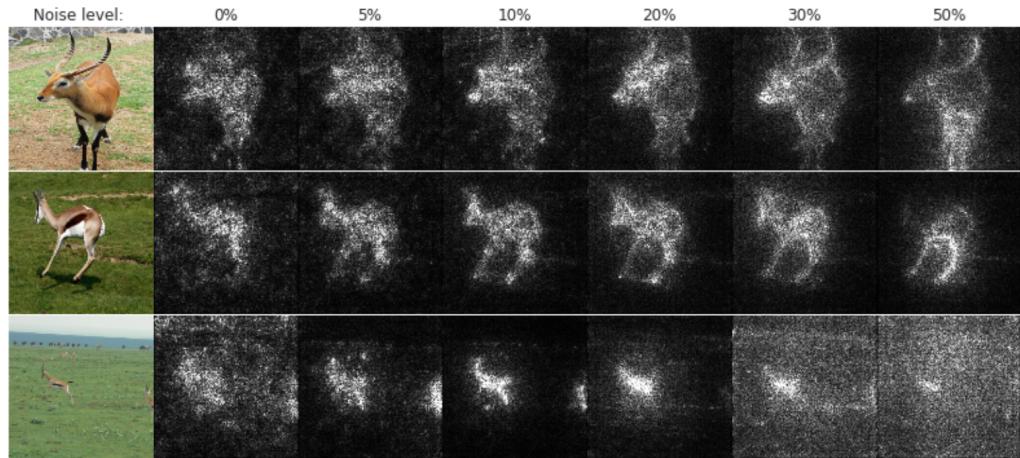
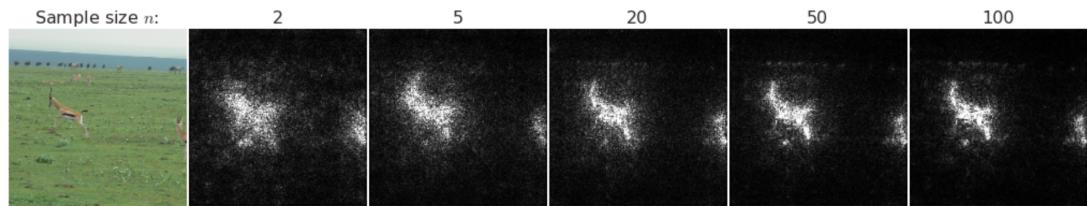


Figure 3. Effect of noise level (columns) on our method for 5 images of the gazelle class in ImageNet (rows). Each sensitivity map is obtained by applying Gaussian noise  $\mathcal{N}(0, \sigma^2)$  to the input pixels for 50 samples, and averaging them. The noise level corresponds to  $\sigma/(x_{max} - x_{min})$ .

- (3) 求梯度次数的影响: 从图中的效果来看, 计算的次数越多越好



- (4) 和其他工作的对比: 作者在对比的过程中, 只是简单地抽取出几张图片对比了一下效果, 说明这个方向缺乏一个 ground truth 来检验各种可解释性方法的好坏

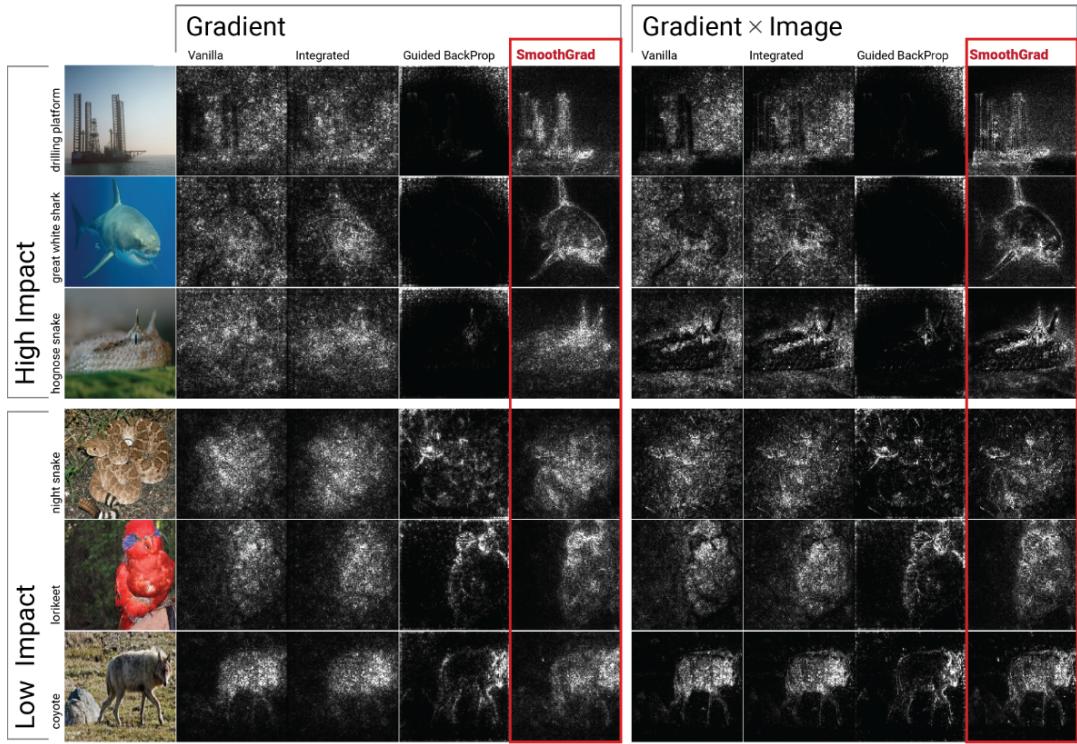


Figure 5. Qualitative evaluation of different methods. First three (last three) rows show examples where applying SMOOTHGRAD had high (low) impact on the quality of sensitivity map.

(5) 最后作者在 MNIST 上面发现, 训练的时候就添加噪声进行训练, 也同样能够达到一定的 "重要性降噪" 的作用:

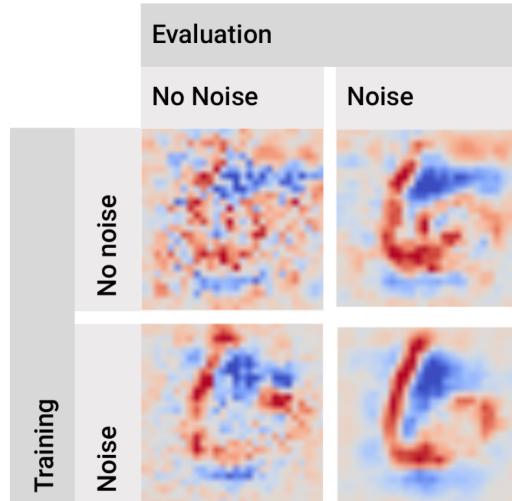


Figure 8. Effect of adding noise during training vs evaluation for MNIST.

## Links

- 论文链接: [Smilkov, Daniel, et al. "Smoothgrad: removing noise by adding noise." ICML \(2017\).](#)
- 论文主页: <https://pair-code.github.io/saliency/>
- Pytorch 实现: [pytorch-smoothgrad](#)
- Tensorflow 实现 (论文源码): [saliency](#)

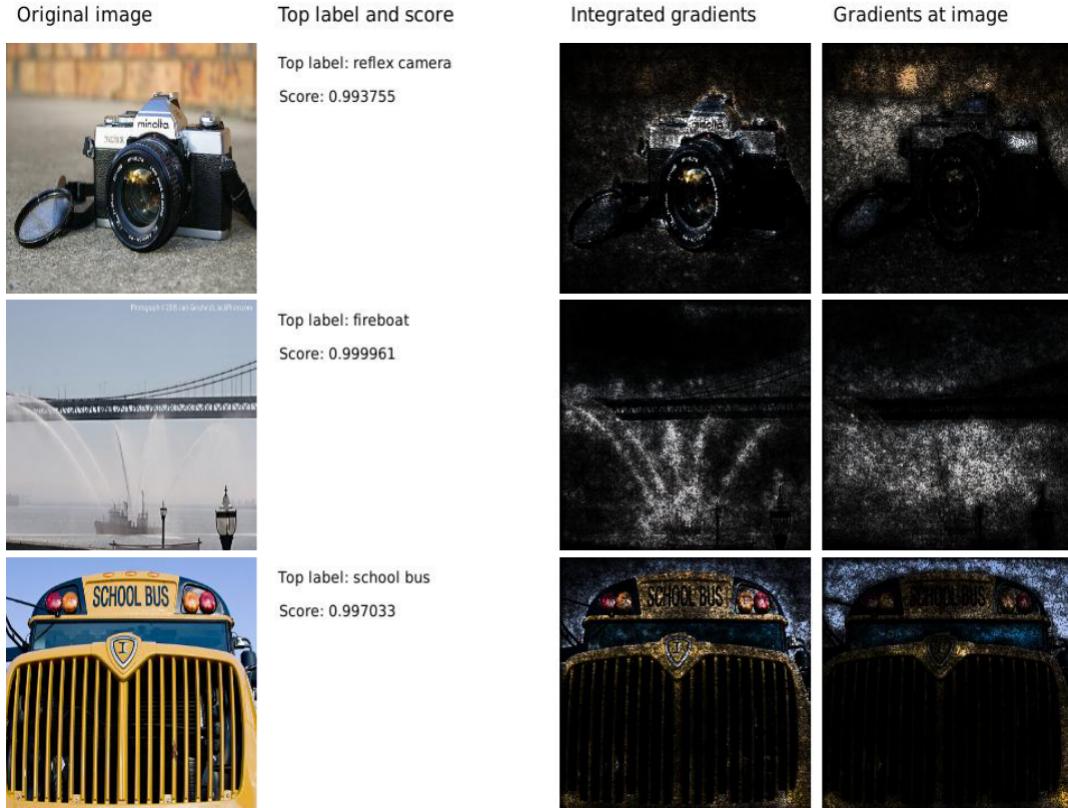
## Axiomatic Attribution for Deep Networks

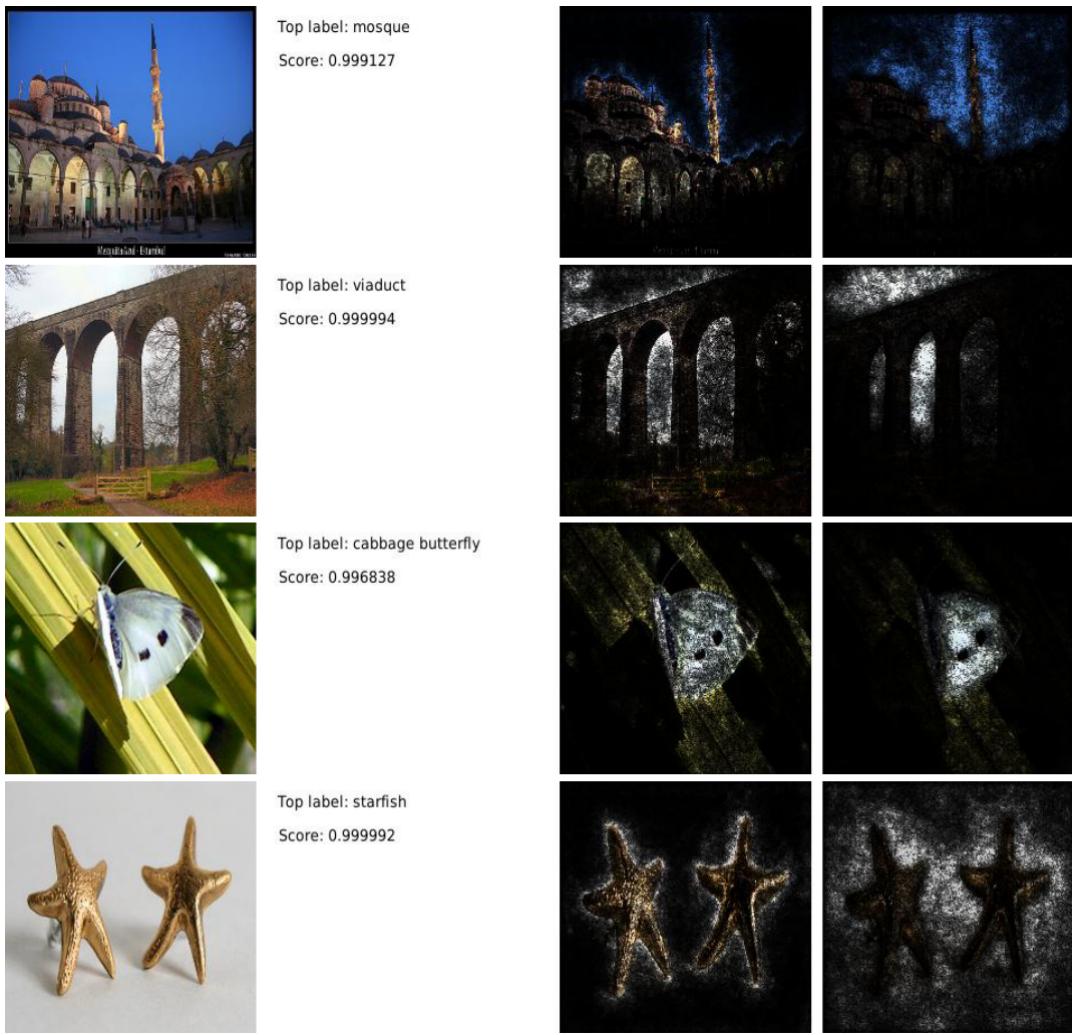
# Contribution

1. 提出了一种基于梯度积分的可解释性方法；
2. 从效果上来看可以在一定程度上解决部分噪点（我认为是一种 梯度饱和问题）；

## Notes

1. 首先简单看下文章在图像分类模型上的效果：





可以看到，作者的方法能够在一定程度上更好地标出更加具体的特征；

2. 两条公理 (Axioms)：作者希望正确的可解释性方法应该至少满足这两条公理

- **Sensitivity**: 如果目标样本和基线样本 (Baseline, 使得目标分类概率为 0 的样本, 本文中用的是全 0 向量), 如果只有一个特征不一样, 导致了两个样本的分类不同, 那么这个特征应该被赋予权重; (梯度和反卷积的方法不满足这一点)
- **Implementation Invariance**: 可解释性方法不应该受到网络内部结构的影响; (LRP 和 DeepLift 方法不满足这一点)

3. Integrated Gradients: 梯度积分可解释性方法

- 理论公式:

The integrated gradient along the  $i^{th}$  dimension for an input  $x$  and baseline  $x'$  is defined as follows. Here,  $\frac{\partial F(x)}{\partial x_i}$  is the gradient of  $F(x)$  along the  $i^{th}$  dimension.

$$\text{IntegratedGrads}_i(x) := (x_i - x'_i) \times \int_{\alpha=0}^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

该公式满足 Completeness:

**Axiom: Completeness.** Integrated gradients satisfy an axiom called *completeness* that the attributions add up to the difference between the output of  $F$  at the input  $x$  and the *baseline*  $x'$ . This axiom is identified as being desirable by Deeplift and LRP. It is a sanity check that the attribu-

用数学公式表达：

**Proposition 1.** If  $F : \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable almost everywhere<sup>1</sup> then

$$\sum_{i=1}^n \text{IntegratedGrads}_i(x) = F(x) - F(x')$$

对于上式的数学证明（参考链接中的博客）：

$$\begin{aligned} \sum_1^n \text{integratedGrads}_i(x) &= \sum_1^n (\gamma_i(1) - \gamma_i(0)) \int_0^1 \frac{\partial F(\gamma)}{\partial \gamma_i} d\alpha \\ &= \sum_1^n \int_0^1 \frac{\partial F(\gamma)}{\partial \gamma_i} \frac{\partial \gamma_i}{\partial \alpha} d\alpha \\ &= \int_0^1 \frac{\partial F(\gamma)}{\partial \gamma} \cdot \frac{\partial \gamma}{\partial \alpha} d\alpha \\ &= \int_0^1 \frac{\partial F(\gamma)}{\partial \alpha} d\alpha \\ &= F(\gamma(1)) - F(\gamma(0)) = F(x) - F(x') \end{aligned}$$

- 具体实现：用等分来近似计算积分

$$\begin{aligned} \text{IntegratedGrads}_i^{approx}(x) &:= \\ (x_i - x'_i) \times \sum_{k=1}^m &\frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m} \end{aligned}$$

## Links

- 论文链接：[Sundararajan M, Taly A, Yan Q. Axiomatic attribution for deep networks\[C\]//International Conference on Machine Learning. PMLR, 2017: 3319-3328.](#)
- 参考博客：[论文笔记：Axiomatic Attribution for Deep Networks](#)
- 论文代码：[Integrated Gradients](#)

# Visualizing and Understanding Neural Machine Translation

上过刘洋老师的 NLP 课后，一直很崇拜刘洋老师！😊😊😊

## Contribution

- 将 LRP (Layer-wise Relevance Propagation) 方法应用到“基于注意力机制的机器翻译任务”中，用于判断不同层不同神经元（或称为隐藏状态）对结果的贡献度；

## Notes

原文在各个细节方面都讲得非常清楚，并且逻辑清晰、言语流畅，因此十分推荐直接阅读原文，下面只是对原文的一些摘要；

- 基于注意力机制的机器翻译任务：

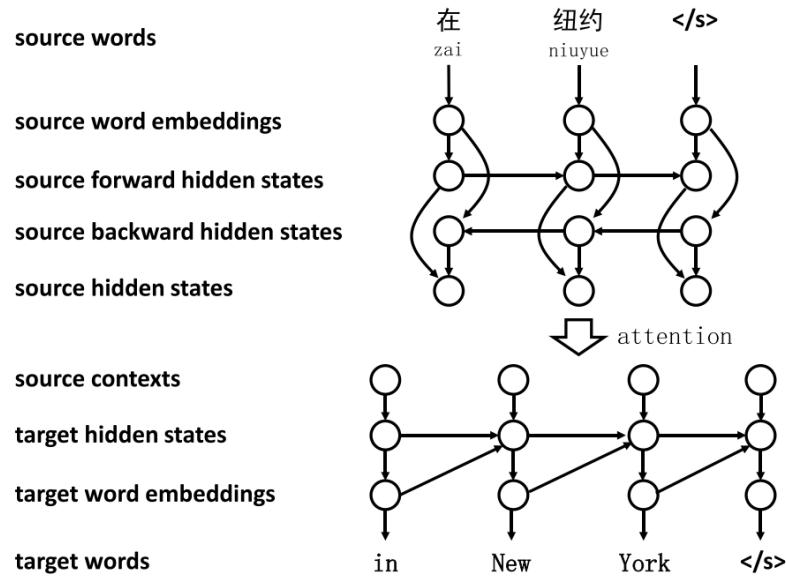


Figure 1: The attention-based encoder-decoder architecture for neural machine translation (Bahdanau et al., 2015).

2. LRP 算法的简单示例：（文章内容有些多，但其实如果你知道 LRP 算法是怎么做的话，这篇文章只是对这个方法的应用）

以一个简单的前向神经网络举例，网络结构如下：

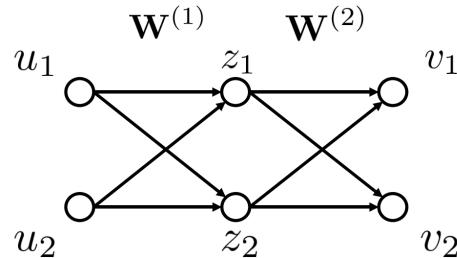


Figure 3: A simple feed-forward network for illustrating layer-wise relevance propagation (Bach et al., 2015).

现在我们关注输入层和隐藏层各神经元对输出神经元  $v_1$  的影响，首先我们计算隐藏层的影响因子：

$$r_{z_1 \leftarrow v_1} = \frac{\mathbf{W}_{1,1}^{(2)} z_1}{\mathbf{W}_{1,1}^{(2)} z_1 + \mathbf{W}_{2,1}^{(2)} z_2} v_1$$

$$r_{z_2 \leftarrow v_1} = \frac{\mathbf{W}_{2,1}^{(2)} z_2}{\mathbf{W}_{1,1}^{(2)} z_1 + \mathbf{W}_{2,1}^{(2)} z_2} v_1$$

然后我们计算输入层的影响因子：

$$\begin{aligned}
r_{u_1 \leftarrow v_1} &= \frac{\mathbf{W}_{1,1}^{(1)} u_1}{\mathbf{W}_{1,1}^{(1)} u_1 + \mathbf{W}_{2,1}^{(1)} u_2} r_{z_1 \leftarrow v_1} + \\
&\quad \frac{\mathbf{W}_{1,2}^{(1)} u_1}{\mathbf{W}_{1,2}^{(1)} u_1 + \mathbf{W}_{2,2}^{(1)} u_2} r_{z_2 \leftarrow v_1} \\
r_{u_2 \leftarrow v_1} &= \frac{\mathbf{W}_{2,1}^{(1)} u_2}{\mathbf{W}_{1,1}^{(1)} u_1 + \mathbf{W}_{2,1}^{(1)} u_2} r_{z_1 \leftarrow v_1} + \\
&\quad \frac{\mathbf{W}_{2,2}^{(1)} u_2}{\mathbf{W}_{1,2}^{(1)} u_1 + \mathbf{W}_{2,2}^{(1)} u_2} r_{z_2 \leftarrow v_1}
\end{aligned}$$

3. LRP 算法定义：（从算法的定义上来看，算法并不需要依赖于可导的梯度计算）

**Input:** A neural network  $G$  for a sentence pair and a set of hidden states to be visualized  $\mathcal{V}$ .  
**Output:** Vector-level relevance set  $\mathcal{R}$ .

```

1 for  $u \in G$  in a forward topological order do
2   | for  $v \in \text{OUT}(u)$  do
3     |   | calculating weight ratios  $w_{u \rightarrow v}$ ;
4   | end
5 end
6 for  $v \in \mathcal{V}$  do
7   | for  $v \in \mathcal{V}$  do
8     |   |  $r_{v \leftarrow v} = v$ ; // initializing neuron-level relevance
9   | end
10  | for  $u \in G$  in a backward topological order do
11    |   |  $r_{u \leftarrow v} = \sum_{z \in \text{OUT}(u)} w_{u \rightarrow z} r_{z \leftarrow v}$ ; // calculating neuron-level relevance
12  | end
13  | for  $u \in \mathcal{C}(v)$  do
14    |   |  $R_{u \leftarrow v} = \sum_{u' \in u} \sum_{v' \in v} r_{u' \leftarrow v'}$ ; // calculating vector-level relevance
15    |   |  $\mathcal{R} = \mathcal{R} \cup \{R_{u \leftarrow v}\}$ ; // Update vector-level relevance set
16  | end
17 end

```

**Algorithm 1:** Layer-wise relevance propagation for neural machine translation.

其中， $v \in \text{OUT}(u)$  指在神经网络中神经元  $u$  指向神经元  $v$ ； $w_{u \rightarrow z}$  的定义如下（在 LRP 原文中，作者发现这个相关系数和网络的激活函数是无关的，所以下面均没有考虑神经元的激活函数）

- 如果是形如  $\mathbf{v} = \mathbf{W}\mathbf{u}$  这样的矩阵相乘，则

$$w_{u \rightarrow v} = \frac{\mathbf{W}_{u,v} u}{\sum_{u' \in \text{IN}(v)} \mathbf{W}_{u',v} u'}$$

- 如果是形如  $\mathbf{v} = \mathbf{u}_1 \circ \mathbf{u}_2$  的向量点积，则

$$w_{u \rightarrow v} = \frac{u}{\sum_{u' \in \text{IN}(v)} u'}$$

- 如果是形如  $v = \max\{u_1, u_2\}$  的求最值，则

$$w_{u \rightarrow v} = \begin{cases} 1 & \text{if } u = \max_{u' \in \text{IN}(v)} \{u'\} \\ 0 & \text{otherwise} \end{cases}$$

其中， $u' \in \text{IN}(v)$  指在神经元中神经元  $u'$  指向神经元  $v$ ；

4. 实验：从实验的结果来看，LRP 方法不仅能够得到和 Attention 可视化方法相似的结果，并且能够分析更多的隐藏层内容。（正例是对正常翻译结果的分析，而反例则是机器翻译中常出现错误翻译问题的举例）

- （正例）Source Side，即 Encoder 侧的影响因子可视化：

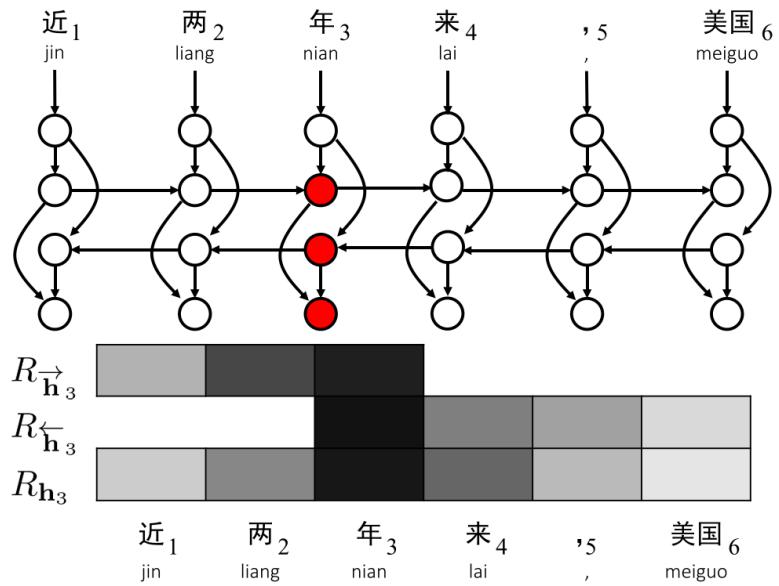


Figure 4: Visualizing source hidden states for a source content word “nian” (years).

- (正例) Target Side, 即 Decoder 侧的印象因子可视化:

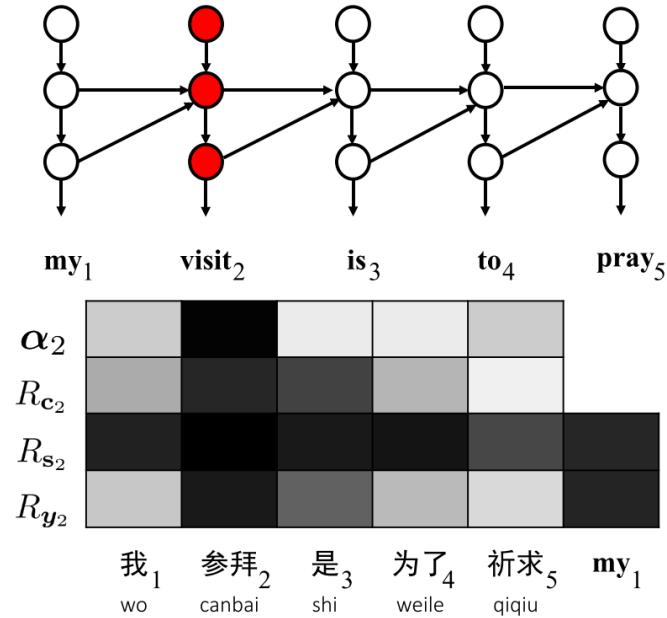


Figure 5: Visualizing target hidden states for a target content word “visit”.

- (反例) Word Omission, 即过早结束翻译:

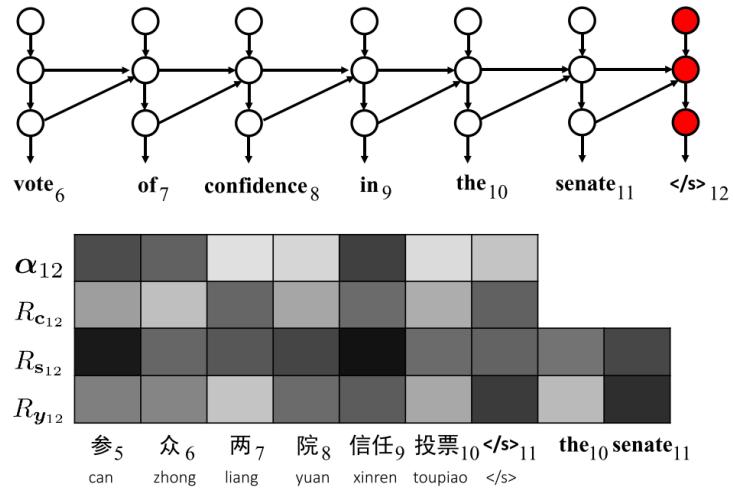


Figure 7: Analyzing translation error: word omission. The 6-th source word “zhong” is untranslated incorrectly.

- (反例) Word Repetition, 即重复翻译;

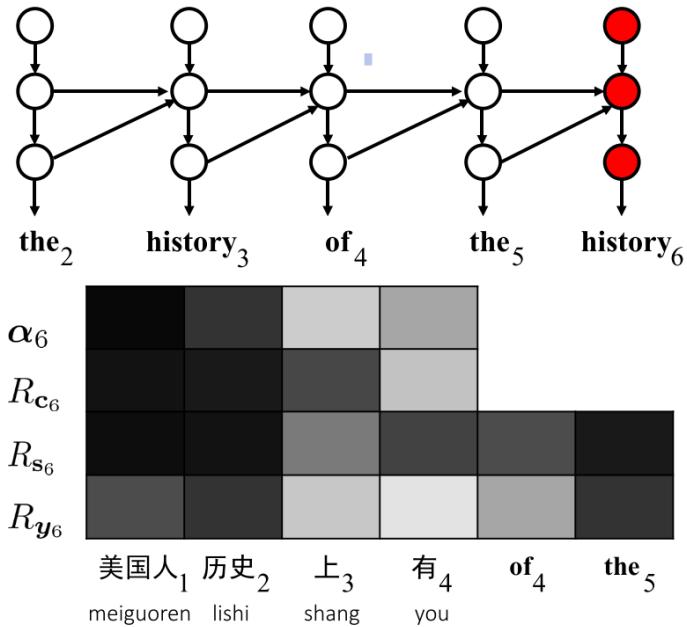


Figure 8: Analyzing translation error: word repetition. The target word “history” occurs twice in the translation incorrectly.

- (反例) Unrelated Words, 即翻译出了完全无关的词汇;

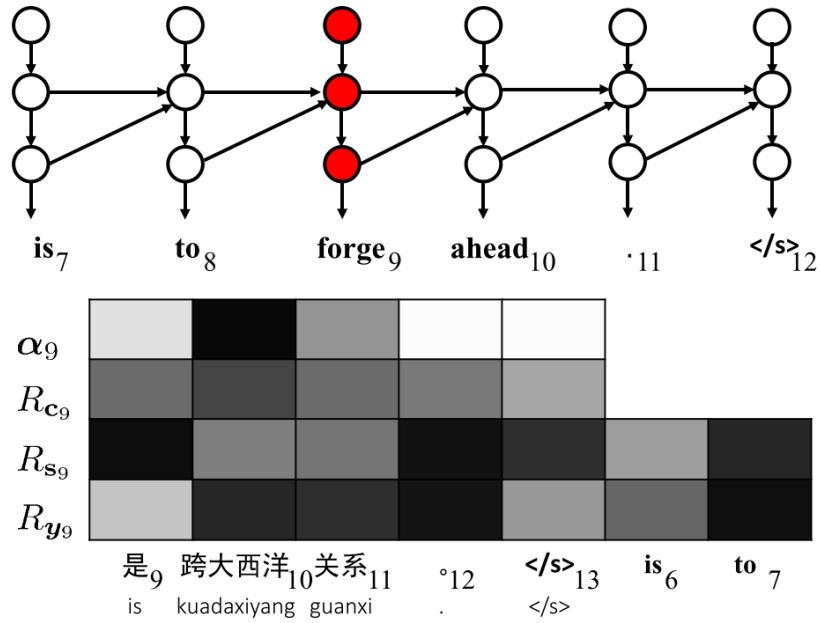


Figure 9: Analyzing translation error: unrelated words. The 9-th target word “forge” is totally unrelated to the source sentence.

- (反例) Negation Reversion, 即翻译时把否定词遗漏了; (文章的这个地方, 我觉得是问题最大的, 翻译过程中把“不”给遗漏了, 不一定分析“about”这个词, 作者对这个词的分析位置的选择, 我觉得纯粹是从中文的角度来看问题的, 因为“谈(talk)”后面跟的是“不”, 他就来分析这个“about”)

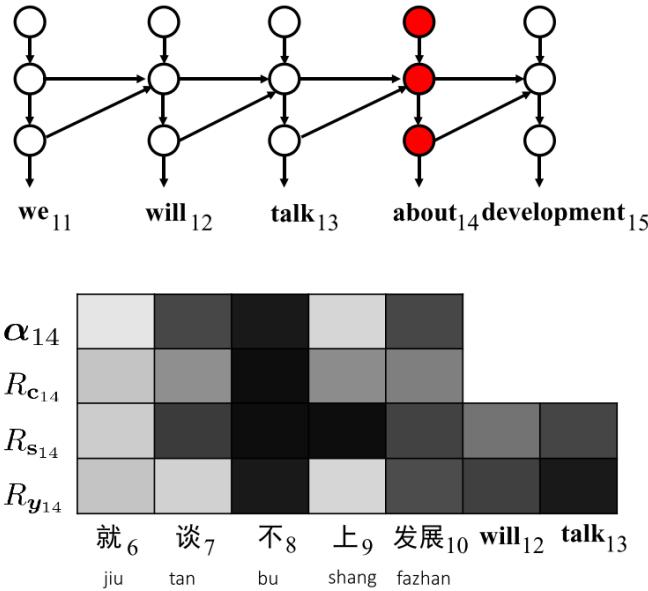


Figure 10: Analyzing translation error: negation. The 8-th negation source word “bu” (not) is not translated.

- (反例) Extra Words, 即翻译中出现了多余的词;

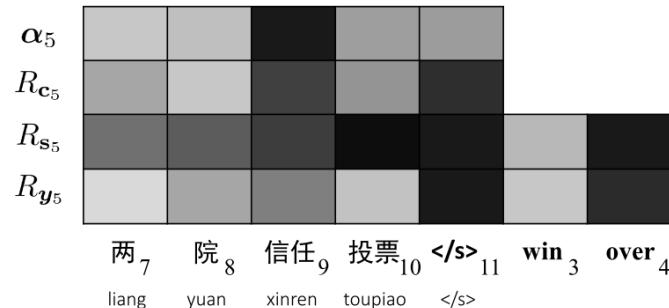
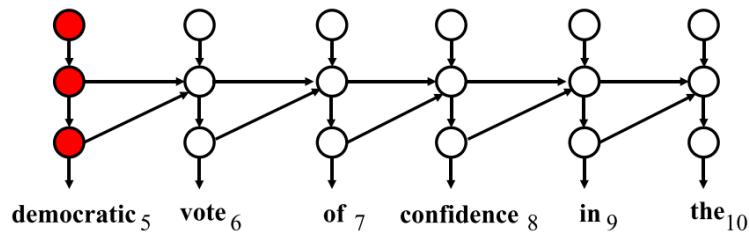


Figure 11: Analyzing translation error: extra word. The 5-th target word “democratic” is an extra word.

## Links

- 论文链接: [Ding Y, Liu Y, Luan H, et al. Visualizing and understanding neural machine translation\[C\]//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics \(Volume 1: Long Papers\). 2017: 1150-1159.](#)

## SHAP - SHapley Additive exPlanation

SHAP 提出的是一种特征重要性的度量方式，在具体的使用上它可以结合已有的可解释性方法（LIME、GRAD、DeepLIFT等）一起运行，所以该部分不仅仅只关注其中某篇论文，而是关注 SHAP 这个方法；

因为 SHAP 实在是太难理解了，纯靠论文我实在是无法理解，所以下面我参考了大量网上的博客，如果有什么不清楚的地方，还是建议去读一读博客的原文和论文；

## Contribution

- 给出了一种具有理论（Shapley）保证的黑盒模型可解释性方法；
- 统一了已有的集中可解释性方法；

## Notes

### SHAP 的基本运作原理

参考博客: [SHAP Values Explained Exactly How You Wished Someone Explained to You](#)

中文翻译: [SHAP 值的解释，以一种你期望的方式解释给你听](#) (这个翻译某些部分有点糟糕，但大致能够看懂，建议直接看英文原文)

## Game Theory and Machine Learning

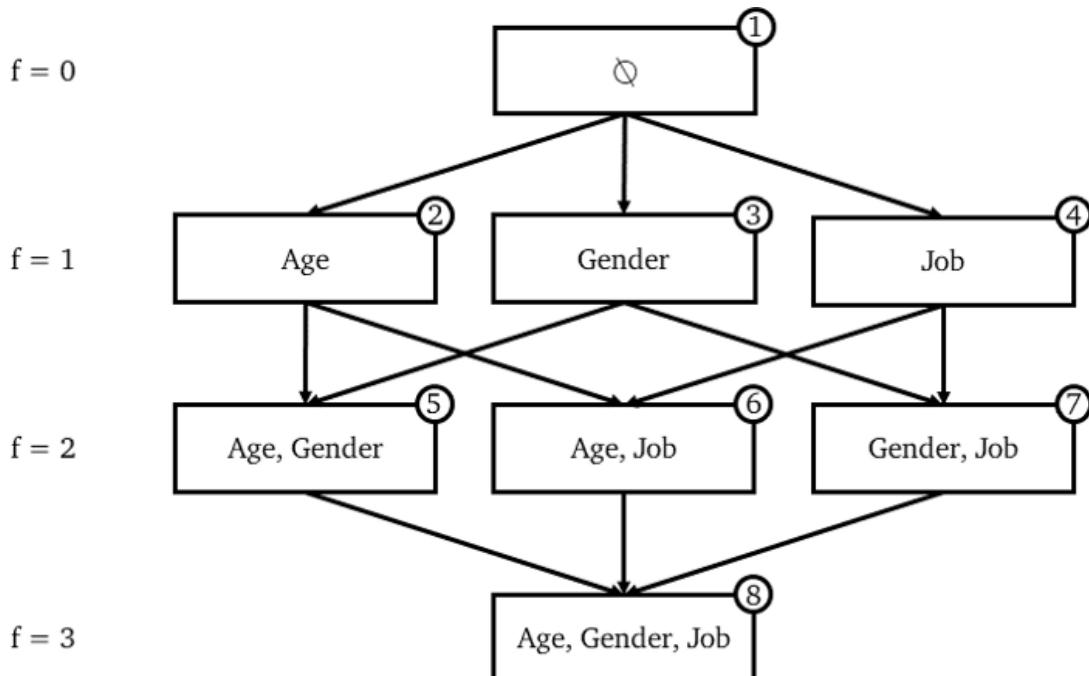
- Shapley Values, 是一个博弈论的概念, SHAP 方法本质上还是算这个值;
- 博弈论中, 涉及到 “Game” (游戏) 和 “Players” (玩家) 两个概念。这两个概念和我们要解决的“模型可解释性”问题的对应关系为:
  - Game 对应的是 **模型的输出值**; (作者特别强调了, “**One Game one observation**”, “一场游戏 对应着一个样本输入到模型中得到的输出值”)
  - Players 对应的是 **模型的特征**;

☆故 Shapley Values 和 SHAP 方法的对应关系为:

- Shapley Values 是用来度量每个 “Player” 对 “Game” 的贡献的;
- SHAP 是用来度量每个特征对模型输出值的贡献的;

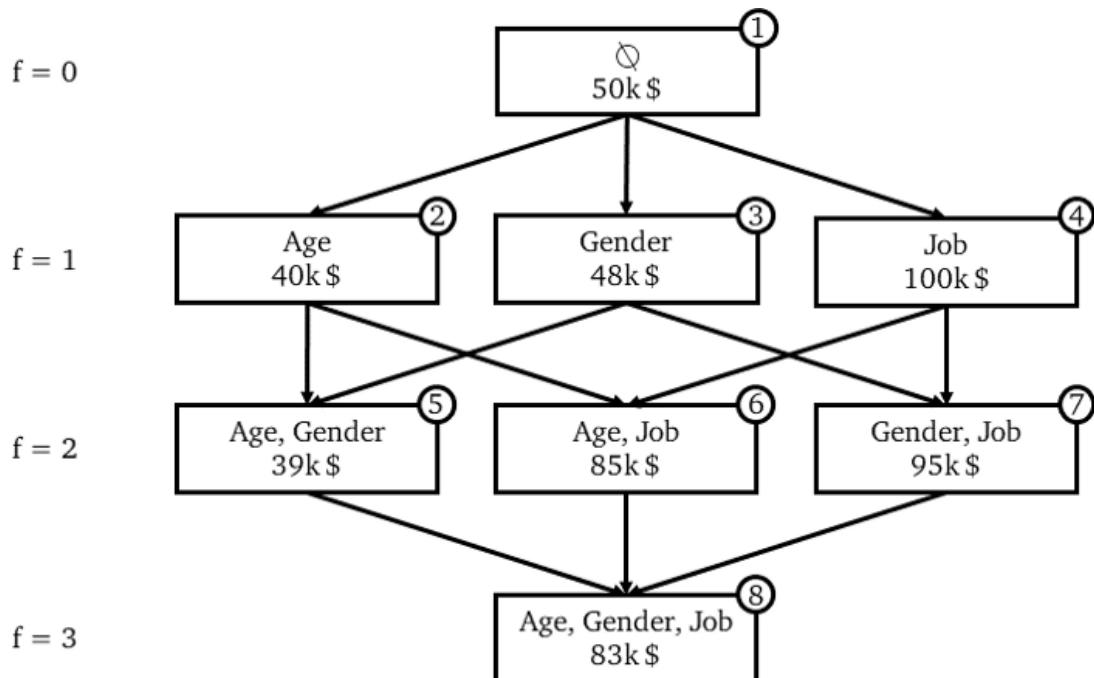
### A Power Set of Features

- 作者以 “用年龄 (Age) 、性别 (Gender) 和工作 (Job) 来预测一个人的收入” 进行举例  
(SHAP 是一中黑盒的可解释性方法, 故这里我们并不需要关心实际的模型是个线性模型、决策树模型还是其他更加复杂的模型) ;
- Shapley Values 的基本思想: **Each possible combination (or coalition) of players should be considered to determine the importance of a single player.** 翻译到模型可解释性方向的话, 就是 计算单个特征的重要性时, 我们需要考虑全部特征的可能组合;
- Power Set (幂集) : 即全部特征的、所有可能的组合, 可以用树的形式来表示, 如下所示;



其中, 每个节点代表一个 **特征联盟** (A coalition of features), 每一条边表示 (上面的特征联盟中) **不存在的特征**。 $f$  指的是某个特征联盟中特征的数量,  $F$  指的是全部特征的数量。

☆SHAP 在解释模型时, 需要为每个特征联盟训练一个 **独立的预测模型**, 即在这里, SHAP 需要训练  $2^F = 8$  个模型 (这些模型的架构、超参数的设定和他们的训练数据都是一致的, 唯一不同的是他们考虑的特征是不一样的) 。作者举例, 我们已经训练得到了 8 个独立的预测模型, 他们对某个输入  $x_0$  在模型中的输出值都做了相应的预测, 如下图所示:



△ 针对上面的 **独立的预测模型**, 列出可能存在的两种不同观点:

- 对于每个节点, 采用的是原始的我们要解释的模型, 而不需要对每个节点训练一个模型;
- **对于每个节点, 需要用相同的方法训练一个独立的模型;**

以上两种观点, 我认为第二个是正确的; 但是当 SHAP 和其他可解释方法相结合时, 可以简化这个过程 (即可以不再单独训练每个模型) ;

#### Marginal Contributions of a Feature

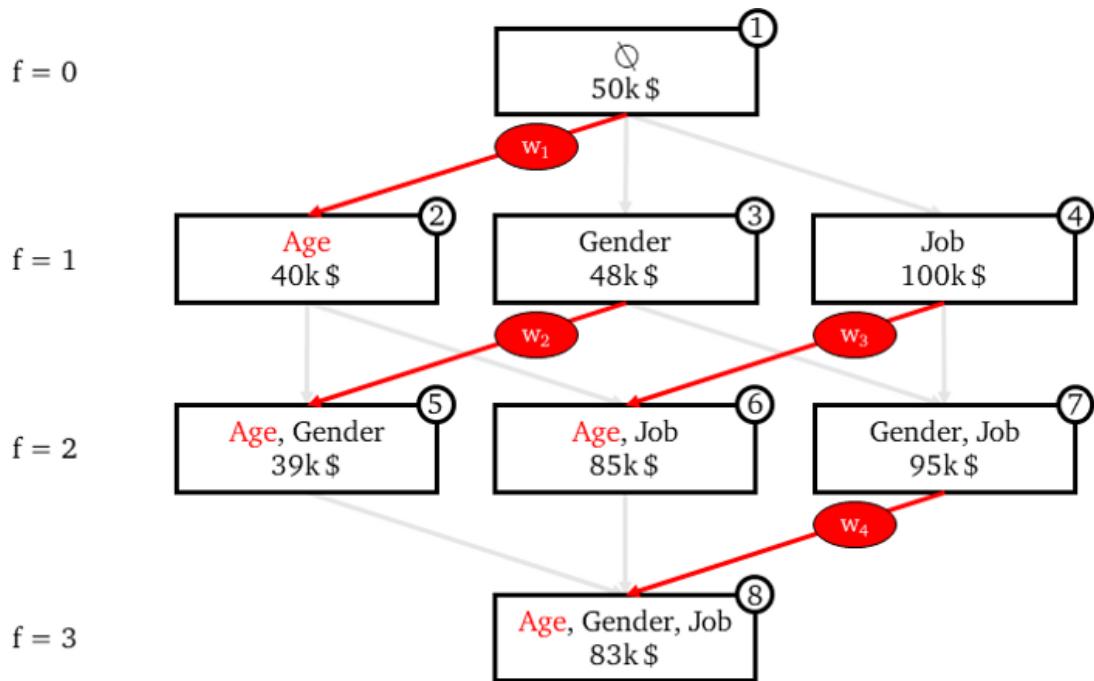
- 如上所述, **两个相连的节点之间只有一个特征的差异**, (前面还指出每个特征联盟的预测模型都使用的相同参数等), 故**两个相连的模型的预测值之间的差异也主要是这个差异特征导致的**; 我们把这种预测值之间的差异称为 **该差异特征的“Marginal Contribution” (边缘贡献)** ; 为了防止我理解的有偏差, 这里给出原文截图:

As seen above, two nodes connected by an edge differ for just one feature, in the sense that the bottom one has exactly the same features of the upper one plus an additional feature that the upper one did not have. Therefore, **the gap between the predictions of two connected nodes can be imputed to the effect of that additional feature. This is called “marginal contribution” of a feature.**

- 举个例子, 对于节点 ①, 这个模型它不考虑任何特征, 所以这个模型给出的就是 (训练集的) 平均收入水平 50k \$; 再看节点 ②, 这个模型它只考虑样本  $x_0$  的年龄 (Age) 特征, 它给出的预测结果是 40k \$; 这说明, 当我们考虑了  $x_0$  的年龄特征后, 模型的预测值下降了 10k \$; 我们把这种差异称为 年龄 (Age) 的边缘贡献 (之一) , 用公式表示如下:

$$MC_{Age,\{Age\}} = Predict_{\{Age\}}(x_0) - Predict_{\emptyset}(x_0) = 40k \$ - 50k \$ = -10k \$$$

下图中画出了年龄 (Age) 的全部边缘贡献:



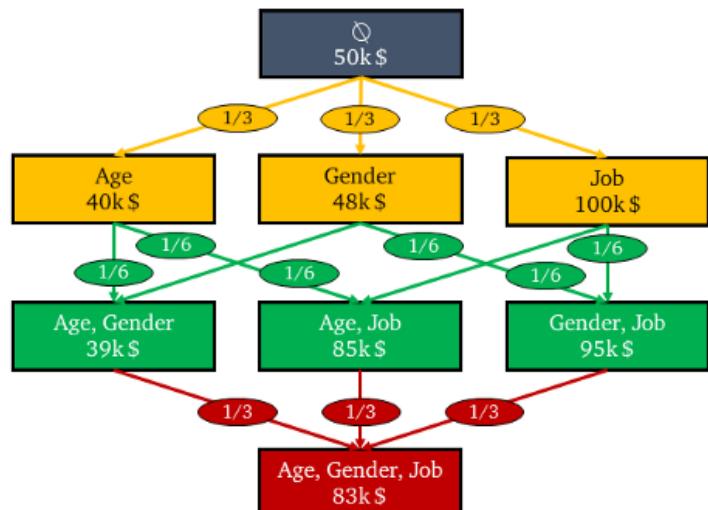
最终，SHAP 对这些边缘贡献进行 **加权求和**（权重的大小由下一小节解释）：

$$\begin{aligned}
 SHAP_{Age}(x_0) = & w_1 \times MC_{Age,\{Age\}}(x_0) \\
 & + w_2 \times MC_{Age,\{Age,Gender\}}(x_0) \\
 & + w_3 \times MC_{Age,\{Age,Job\}}(x_0) \quad \text{where } w_1 + w_2 + w_3 + w_4 = 1 \\
 & + w_4 \times MC_{Age,\{Age,Gender,Job\}}(x_0)
 \end{aligned}$$

### Weighting the Marginal Contributions

- 以上面的为例，我们直接给出每个边缘贡献的权重，如下图所示：

|       | N. of Nodes<br>$\binom{F}{f}$ | N. of Edges<br>$f \times \binom{F}{f}$ |
|-------|-------------------------------|--|
| f = 0 | 1                             |  |
| f = 1 |                               | 3                                      |
|       | 3                             |  |
| f = 2 |                               | 6                                      |
|       | 3                             |  |
| f = 3 |                               | 3                                      |
|       | 1                             |  |
| Sum   | $2^F = 8$                     | $F \times 2^{F-1} = 12$                |



简单来说，每个边缘贡献的权重 等于 每层边缘贡献数量的倒数；

- 数学公式对上面的规律进行表示：

$$weight = f \times \binom{F}{f}$$

- 现在，我们可以求出最终的 SHAP 值：

$$\begin{aligned}
SHAP_{Age}(x_0) &= [1 \times \binom{3}{1}]^{-1} \times MC_{Age,\{Age\}}(x_0) \\
&\quad + [2 \times \binom{3}{2}]^{-1} \times MC_{Age,\{Age,Gender\}}(x_0) \\
&\quad + [2 \times \binom{3}{2}]^{-1} \times MC_{Age,\{Age,Job\}}(x_0) \\
&\quad + [3 \times \binom{3}{3}]^{-1} \times MC_{Age,\{Age,Gender,Job\}}(x_0) \\
&= \frac{1}{3} \times (-10k \$) + \frac{1}{6} \times (-9k \$) + \frac{1}{6} \times (-15k \$) + \frac{1}{3} \times (-12k \$) \\
&= -11.33k \$
\end{aligned}$$

## Wrapping it up

- 最后，我们就得到了 SHAP 文章中的公式（现在就很好理解这个公式了）：

$$SHAP_{feature}(x) = \sum_{set: feature \in set} [|set| \times \binom{F}{|set|}]^{-1} [Predict_{set}(x) - Predict_{set \setminus feature}(x)]$$

- 相应的，我们可以计算得到各个特征的 SHAP 值：

$$SHAP_{Age}(x_0) = -11.33k \$ SHAP_{Gender}(x_0) = -2.33k \$ SHAP_{Job}(x_0) = +46.66k \$$$

可以看到，把这些 SHAP 值全部加起来，正好等于 the full model（最下面的考虑全部特征的模型）减去 the null model（最上面的不考虑任何特征的模型）的预测值的差；这也正好是 **SHAP 方法的一个特点**，也是它取名的原由，为了不造成更多歧义，直接粘原文：

This is a fundamental characteristic of SHAP values: **summing the SHAP values of each feature of a given observation yields the difference between the prediction of the model and the null model** (or its logistic function, as we have seen [here](#)). This is actually the reason for their name: SHapley Additive exPlanations.

- 这里，虽然我们似乎已经做完了 SHAP 的全部步骤，但是在实际应用中，我们考虑的特征是非常多的，可以看到我们上面还要分别训练  $2^F$  个模型，所以接下来的任务，就是**如何在拥有大量特征的情况下，用近似的方法来求解 SHAP 值**，论文原文的话如下所示：

The exact computation of SHAP values is challenging. However, by combining insights from current additive feature attribution methods, we can approximate them. We describe two model-agnostic approximation methods, one that is already known (Shapley sampling values) and another that is novel (Kernel SHAP). We also describe four model-type-specific approximation methods, two of which are novel (Max SHAP, Deep SHAP). When using these methods, **feature independence and model linearity** are two optional assumptions simplifying the computation of the expected values

## Kernel SHAP (Linear LIME + Shapley Values)

参考博客：[Complete SHAP tutorial for model explanation Part 3. KernelSHAP](#)（建议优先看这篇博客）

参考博客：[5.10.2 KernelSHAP](#)

第二篇博客的中文翻译：[模型解释-SHAP Value的简单介绍](#)（一样，中文翻译一般都挺坑的）

- 符号表示

为了让下面的内容更容易看懂，我们得先解释一下一些符号的含义：

- $x$  表示模型的输入， $f(x)$  表示模型的输出；
- $z'$  表示简化后的**特征域**（如：图像中我们会把模型的输入  $x$  切成小的超像素块域  $z'$ ）， $M$  表示特征的数量， $h_x(z')$  表示将特征域  $z'$  映射到原输入域  $x$ （注意：这个映射是一个 **one-to-many mapping**）；下面展示论文中的描述：

Different types of  $h_x$  mappings are used for different input spaces. For bag of words text features,  $h_x$  converts a vector of 1's or 0's (present or not) into the original word count if the simplified input is one, or zero if the simplified input is zero. For images,  $h_x$  treats the image as a set of super pixels; it then maps 1 to leaving the super pixel as its original value and 0 to replacing the super pixel with an average of neighboring pixels (this is meant to represent being missing).

- $g$  表示目标模型的替代解释模型;
- $\pi_x(z')$  表示为样本  $z'$  的权重;

- LIME 可解释方法

LIME 的目标是最小化如下**目标函数**:

$$\xi = \arg \min_{g \in \mathcal{G}} L(f, g, \pi_x(z')) + \Omega(g)$$

即, LIME 希望最终得到的线性回归模型在采样点  $z'$  上的加权误差尽可能的小, 并且根据样本的修改程度来确定不同样本的权重  $\pi_x(z')$ , 使用  $\Omega(g)$  来限制回归模型的复杂度;

- ☆ Kernel SHAP 运行步骤

1. 采样特征域样本  $z'$ , 一般地, 我们需要采样  $2^M$  个特征域样本;

2. 计算模型在采样样本上的预测结果  $f(h_x(z'))$ ;

因为上面提到, 映射变换是一个一对多的变换, 所以这里  $f(h_x(z')) = E[f(h(z'))]$ ;  
 ; (是否会为了简化计算, 把这个一对多的映射变成一个一对一的映射? 这种映射可能还存在一个问题, 就是说它的前提是每个特征需要相互独立的, 但是在实际的数据分布中, 并不是这样的; )

3. 用如下 **SHAP Kernel** 计算每个特征域样本的权重:

$$\pi_x(z') = \frac{(M - 1)}{\binom{M}{|z'|} |z'| (M - |z'|)}$$

4. 借助 LIME, 构造**线性回归模型**:

$$g(z') = \phi_0 + \sum_{j=1}^M \phi_j z'_j$$

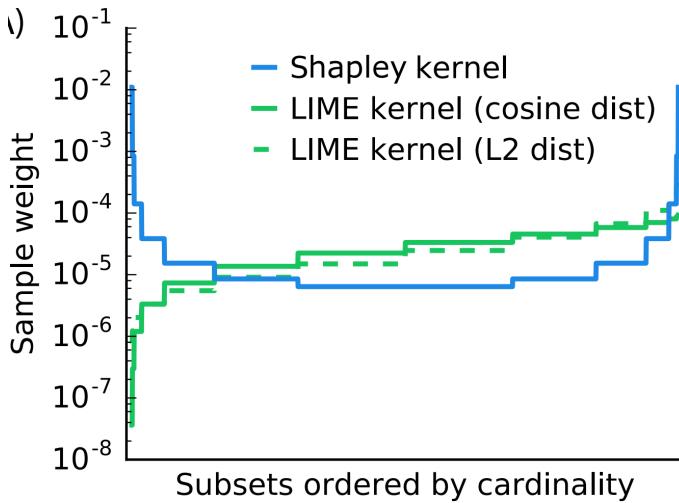
去除原目标函数中的复杂度约束项  $\Omega(g)$ , 并将新的 SHAP Kernel 代入公式中, 得到**新的目标函数**:

$$\xi = \arg \min_{g \in \mathcal{G}} L(f, g, \pi_x(z')) L(f, g, \pi_x(z')) = \sum_{z' \in Z} [f(h_x(z')) - g(z')] \cdot \pi_x(z')$$

5. 拟合线性回归模型, 得到的  $\phi_j$  即为 **Shapley Values 的估计值**; (论文中证明了, 这种方法可以保证得到 Shapley Values 的估计值, 具体的证明细节就不再探讨)

- 理解 SHAP Kernel

为了理解 SHAP Kernel 这个核函数, 我们对比一下其和 LIME 原先使用的启发式核函数的区别, 如下图所示:



可以看到，原先的启发式核函数保持的联盟特征越多，其权重越大；而 SHAP Kernel 中，联盟中的特征很多或者很少时，联盟的权重相对更高；这造成的影响是，最后的回归模型对于小联盟或者大联盟的拟合越好；给出这一段的博客原文描述：

The plot indicates both small coalitions (few 1's in  $\mathbf{z}'$ ) and large coalitions (many 1's in  $\mathbf{z}'$ ) get the largest weights, others get small weights.

So this type of weight distributions cause regression focus more on predicting small coalitions and largest coalitions, and overlook other coalitions. In other words, regression will be more accurate on small coalitions and largest coalitions.

- 理解 Kernel SHAP 方法（是我个人的理解，谨慎参考）

- Kernel SHAP 可解释方法，可以简单地认为是 LIME 方法的一个变种，因为我们是在 LIME 方法上对目标函数和样本权重公式做了相应的修改，最后还是拟合了一个线性回归模型；
- 但是，我认为这种方法可能并没有 LIME 方法来的好，因为 我可能更希望通过样本间的距离来赋值我的样本权重；
- Kernel SHAP 运行还是比较耗时，另外 Tree SHAP 在效率上得到了提高；
- Kernel SHAP 和已有的 Shapley Values 算法的区别是什么？是可以不再独立计算每一个联盟模型了吗？

- 代码理解：

代码库：<https://github.com/slundberg/shap>

Debug 的脚本：[ImageNet VGG16 Model with Keras](#)

首先，简单讲一下这个脚本的逻辑：

```

1 # 首先，我们获取一张图片
2 file = "dataset/ILSVRC2012_val_00001419.JPG"
3 img = image.load_img(file, target_size=(224, 224))
4 img_orig = image.img_to_array(img)
5 # 然后，我们用 skimage.segmentation 库对图片进行像素块划分
6 segments_slic = slic(img, n_segments=50, compactness=30, sigma=3)
7 # 加载一个 keras 中预训练的 vgg16 模型
8 model = VGG16()
9 # 对模型的预测进行简单的包装，即我们在解释模型结果的过程中，是针对像素块的0/1问题，如果是0，那就像素全部置为白色255；如果是1，就保留原来的像素值
10 def mask_image(zs, segmentation, image, background=None):
11     if background is None:
12         background = image.mean((0, 1))

```

```

13     out = np.zeros((zs.shape[0], image.shape[0], image.shape[1],
14                     image.shape[2]))
15     for i in range(zs.shape[0]):
16         out[i, :, :, :] = image
17         for j in range(zs.shape[1]):
18             if zs[i, j] == 0:
19                 out[i][segmentation == j, :] = background
20     return out
21 def f(z):
22     return model.predict(preprocess_input(mask_image(z, segments_slic,
23                                               img_orig, 255)))
24 # 使用可解释方法对其进行解释
25 explainer = shap.KernelExplainer(f, np.zeros((1, 50)))
26 shap_values = explainer.shap_values(np.ones((1, 50)), nsamples=1000)
27 # runs VGG16 1000 times

```

然后，讲一下 `KernelExplainer` 部分的核心代码，在文件 `kernel.py` 中：

```

1 # 主要看 explain 函数
2 # 首先确定每个样本的权重，即 SHAP Kernel
3 num_subset_sizes = np.int(np.ceil((self.M - 1) / 2.0))
4 num_paired_subset_sizes = np.int(np.floor((self.M - 1) / 2.0))
5 weight_vector = np.array([(self.M - 1.0) / (i * (self.M - i)) for i in
6 range(1, num_subset_sizes + 1)])
7 weight_vector[:num_paired_subset_sizes] *= 2
8 weight_vector /= np.sum(weight_vector)
9 # 然后如果足够采样一对对称（特征110和001为对称）的联盟样本时，则根据上面的概率进行
10 # 全采样
11 # for subset_size in range(1, num_subset_sizes + 1):
12 #     ...
13 #     ...
14 # 当无法进行全采样时，则根据相同的概率对样本进行采样
15 while samples_left > 0 and ind_set_pos < len(ind_set):
16     ...
17 # 然后调用 LIME 算法
18 self.run()

```

## Gradient SHAP

- 原论文中，作者并没有提到 Gradient SHAP 这种方法，但是在代码中有这个方法，我们从 [Captum 库中的介绍](#) 中来大概理解一下这个方法：

GradientShap approximates SHAP values by computing the expectations of gradients by randomly sampling from the distribution of baselines/references. It adds white noise to each input sample `n_samples` times, selects a random baseline from baselines' distribution and a random point along the path between the baseline and the input, and computes the gradient of outputs with respect to those selected random points. The final SHAP values represent the expected values of gradients \* (inputs - baselines).

GradientShap makes an assumption that the input features are independent and that the explanation model is linear, meaning that the explanations are modeled through the additive composition of feature effects. Under those assumptions, SHAP value can be approximated as the expectation of gradients that are computed for randomly generated `n_samples` input samples after adding gaussian noise `n_samples` times to each input for different baselines/references.

In some sense it can be viewed as an approximation of integrated gradients by computing the expectations of gradients for different baselines.

Current implementation uses Smoothgrad from NoiseTunnel in order to randomly draw samples from the distribution of baselines, add noise to input samples and compute the expectation (smoothgrad).

再看看 [Shap 库中的介绍](#):

## Deep learning example with GradientExplainer (TensorFlow/Keras/PyTorch models)

Expected gradients combines ideas from [Integrated Gradients](#), SHAP, and [SmoothGrad](#) into a single expected value equation. This allows an entire dataset to be used as the background distribution (as opposed to a single reference value) and allows local smoothing. If we approximate the model with a linear function between each background data sample and the current input to be explained, and we assume the input features are independent then expected gradients will compute approximate SHAP values. In the example below we have explained how the 7th intermediate layer of the VGG16 ImageNet model impacts the output probabilities.

大概可以理解为：用 [Integrated Gradients](#) 和 [SmoothGrad](#) 方法结合，通过拟合梯度的期望值来近似计算 Shapley Value.

- 代码理解：

代码库：<https://github.com/slundberg/shap>

Debug 的脚本：[Explain an Intermediate Layer of VGG16 on ImageNet](#)

同样，看一下这个脚本的逻辑：

```
1 # 加载VGG16模型
2 model = VGG16(weights='imagenet', include_top=True)
3 # 加载一个background数据集，这个数据集要被用来进行随机采样
4 X,y = shap.datasets.imagenet50()
5 # 选择要进行解释的两个样本
6 to_explain = X[[39,41]]
7 # 定义了一个获取第layer层输入的函数
8 def map2layer(X, layer):
9     feed_dict = dict(zip([model.layers[0].input],
10                         [preprocess_input(X.copy())]))
11     return K.get_session().run(model.layers[layer].input, feed_dict)
12 # 实例化一个解释器
13 e = shap.GradientExplainer(
14     model=(model.layers[7].input, model.layers[-1].output), # 因为这里是keras模型，所以用这种形式传入模型，这里是想解释第7层的特征重要性
15     data=map2layer(preprocess_input(X.copy()), 7) # 可以看到，这里把background数据集的第7层的输入全部传入了解释其中，供后面进行采样
16 )
```

```

16 # 对两个样本进行解释
17 shap_values, indexes = e.shap_values(map2layer(to_explain, 7),
ranked_outputs=2)

```

然后，讲一下 `GradientExplainer` 的核心代码，在 `_gradient.py` 文件中：

```

1 # 主要看 _TFGradient.shap_values 方法，代码从250行开始是关键
2 # 采样n个样本点
3 for k in range(nsamples):
4     rind = np.random.choice(self.data[0].shape[0])
5     t = np.random.uniform()
6     for l in range(len(X)):
7         if self.local_smoothing > 0:
8             x = X[l][j] + np.random.randn(*X[l][j].shape) *
9             self.local_smoothing # 可以看到 SmoothGrad 的影子，在原样本点上添加扰动
10        else:
11            x = X[l][j]
12        samples_input[l][k] = t * x + (1 - t) * self.data[l][rind] # 可以看到 Integrated Gradient 的影子，在background数据集中随机找一个点，然后在两个样本点之间随机选择一个点作为采样点
13        samples_delta[l][k] = x - self.data[l][rind] # 这里，可以说是权重，也可以说是 integrated Gradient 方法积分公式的第一项
14
15 # 针对要解释的目标分类计算梯度值
16 find = model_output_ranks[j,i]
17 grads = []
18 for b in range(0, nsamples, self.batch_size):
19     batch = [samples_input[l][b:min(b+batch_size,nsamples)] for l in range(len(X))]
20     grads.append(self.run(self.gradient(find), self.model_inputs,
21                           batch))
22 grad = [np.concatenate([g[l] for g in grads], 0) for l in
23          range(len(X))]
24 # 相乘求期望即可
25 for l in range(len(X)):
26     samples = grad[l] * samples_delta[l]
27     phis[l][j] = samples.mean(0)
28     phi_vars[l][j] = samples.var(0) / np.sqrt(samples.shape[0]) #
29     estimate variance of means

```

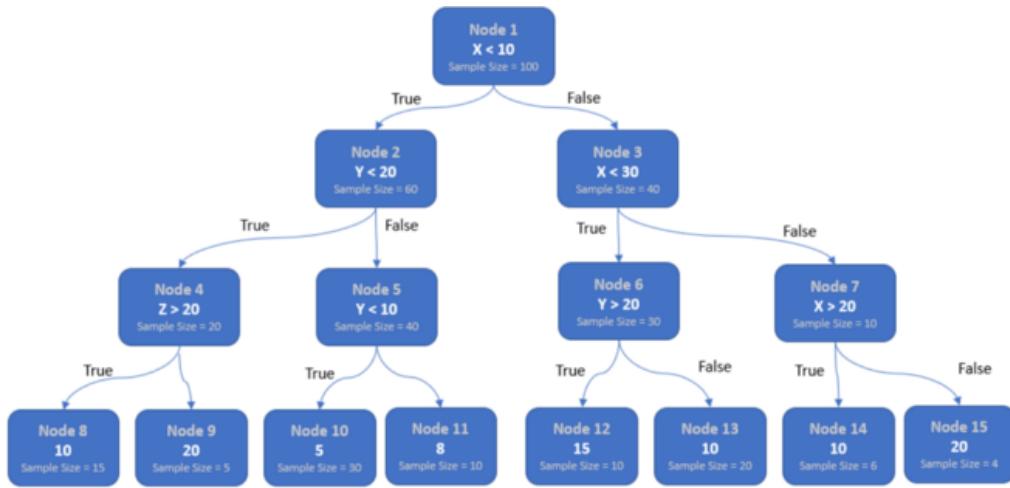
## Tree SHAP

参考博客：[Complete SHAP tutorial for model explanation Part 4. TreeSHAP](#)

参考博客：[SHAP 知识点全汇总](#)（因为不太看得懂，故仅参考了 TREE SHAP 算法复杂度一段）

### Tree Prediction on Feature Missing Instance

- 博客作者首先构造了一棵有 100 个训练样本、3 个特征  $(x, y, z)$  的决策树，如下图所示：



- 下面，我们来看 **如何根据该决策树得到 coalition (联盟) 的预测值**:

◦ 如果我们关注联盟 ( $x = 15$ )，根据决策树路径 ( $1 \rightarrow 3 \rightarrow 6$ )，得到节点 6，但是该节点并不是叶子节点，故我们没法根据节点 6 获取联盟的预测值。所以我们需要根据它的子叶子节点 (12, 13)，求得联盟的预测值：

$$Prediction_{\{x=15\}} = 15 * (10/30) + 10 * (20/30) = 350/30$$

◦ 同理，我们关注联盟 ( $x = 5$ )，我们需要根据节点 4, 5 的子叶子节点 (8, 9, 10, 11)，求得联盟的预测值：

$$Prediction_{\{x=5\}} = 10 * (15/60) + 20 * (5/60) + 5 * (30/60) + 8 * (10/60) = 480/60$$

◦ 最后，我们关注联盟 ( $x = 5, z = 10$ )，我们会直接得到一个叶子节点 (9)，并且我们还需要根据节点 5 的子叶子节点 (10, 11)，求得联盟的预测值：

$$Prediction_{\{x=5,z=10\}} = 20 * (5/45) + 5 * (30/45) + 8 * (10/45) = 330/45$$

- 有了联盟的预测值后，我们就可以 **计算边缘贡献**：

$$MC_{z=10,\{x=5,z=10\}} = Prediction_{\{x=5,z=10\}} - Prediction_{\{x=5\}} = 480/60 - 330/45$$

- 算法复杂度

如果采用上述方法来 **查询联盟的预测值** 的话，显然时间复杂度为树的深度  $O(L)$ ，因为这里我们可能会用到随机森林等算法，所以时间复杂度  $O(TL)$ ，其中  $T$  为树的数量；集合中共有  $2^M$  个联盟，所以**计算 SHAP 值的时间复杂度**为  $O(TL \cdot 2^M)$ ，可以看到这个复杂度和特征的数量指数相关；

☆ 为了解决算法复杂度问题，论文 "[Consistent Individualized Feature Attribution for Tree Ensembles](#)" 提出算法使得该时间复杂度降低到  $O(TLD^2)$ ，当树为一颗平衡二叉树时  $D = \log L$ ；

## Links

- 论文链接：[Lundberg S, Lee S I. A unified approach to interpreting model predictions\[J\]. arXiv preprint arXiv:1705.07874, 2017.](#)
- 论文链接：[Lundberg S M, Erion G G, Lee S I. Consistent individualized feature attribution for tree ensembles\[J\]. arXiv preprint arXiv:1802.03888, 2018.](#)
- 原作者论代码：[slundberg/shap: A game theoretic approach to explain the output of any machine learning model. \(github.com\)](#)
- Pytorch-Captum 库代码：<https://github.com/pytorch/captum>
- API 文档：<https://shap.readthedocs.io/en/latest/>
- SHAP 简单入门博客：[SHAP: Python 的可解释机器学习库](#)

# Beyond Sparsity: Tree Regularization of Deep Models for Interpretability

## Contribution

1. 提出了一个 Tree Regularization 的概念，即决策树的节点高度，希望神经网络能学得更像决策树一样；（从整篇文章来看，好像不太靠谱的样子）
2. 全局地用一个决策树模型来拟合目标神经网络，然后利用决策树模型本身优异地可解释性来解释目标神经网络；

## Links

- 论文链接：[Wu M, Hughes M, Parbhoo S, et al. Beyond sparsity: Tree-based regularization of deep models for interpretability\[C\]//In: Neural Information Processing Systems \(NIPS\) Conference. Transparent and Interpretable Machine Learning in Safety Critical Environments \(TIML\) Workshop, 2017.](#)
- 论文代码：<https://github.com/dtak/tree-regularization-public>

# LEMNA: Explaining Deep Learning based Security Applications

这篇文章，实际上我看了两遍：第一遍研一的时候看的，是文献阅读课，讲得很差，被老板批评了；第二遍研二项目中需要用到一系列的可解释性方法，和大佬一起又重新拜读了一下这篇文章，这次是结合代码来学习的，所以对 LEMNA 这种方法算是又有了新的认识；

## Contribution

1. LEMNA (Local Explanation Method using Nonlinear Approximation)，一种新的黑盒解释方法；
2. Mixture Regression Model 拟合一个局部非线性（分段线性）模型；（用我的话来说，其实是用高斯混合分布筛选扰动数据集）
3. Fused Lasso 限制的临近特征权重之间的差异；

## Notes

1. 先从作者总结的表格来看一下这些已有的可解释性方法：

| Explanation Method         | Support RNN/MLP | Local Non-linear | Support Blackbox | Representative Works                                    |
|----------------------------|-----------------|------------------|------------------|---|
| Whitebox method (forward)  | ●               | ○                | ●                | Occlusion [17, 32, 74, 76], AI <sup>2</sup> [19],       |
| Whitebox method (backward) | ●               | ○                | ○                | Saliency Map [3, 54, 57], Grad-carm [50], DeepLIFT [53] |
| Blackbox method            | ●               | ○                | ●                | LIME [45], SHAP [34], Interpretable Decision Set [31]   |
| Our method LEMNA           | ●               | ●                | ●                | LEMNA   |

Table 1: Design space of explainable machine learning for security applications (●=true; ○=false; □=partially true).

这里简单将现有的工作分为三类：

- White-box method (forward): 通过修改/去除前向传播的一些特征，观察模型输出的变化大小，来确定特征的重要性；
- White-box method (backward): 通过比较梯度的反向传播值的大小、正负，来确定特征的重要性；
- Black-box method: 通过拟合模型的边界来确定特征的重要性；（LEMNA 同样属于这一类）

同时，作者也列出了一些理由，为什么不选择已有的这些工作：

- 是否支持 RNN/MLP，像 CAM 家族的工作都是依赖于卷积层的位置对应关系才得以将热力图回传的，所以这类工作不支持非 CNN 网络；
- 边界是否为非线性的；（挺好奇的是，为什么作者这里认为白盒的方法不支持非线性？）
- 方法是否支持黑盒（这一点，我认为是比较 Weak）；

2. LEMNA 的方法设计：主要分为两块，一个是 Fused Lasso，一个是 Mixture Regression Model；

- Fused Lasso：作者的解释是，为了使得 LEMNA 方法将相关的（文中用的 relevant）或者是靠近的（文中用的 adjacent）特征关联起来，来生成解释。换成我的话说，就是让相邻的（和 relevant 没有一点关系）特征的权重尽可能得相近，看公式便一清二楚；

在 LIME 等前面的可解释性方法中，是使用最大似然估计 MLE 的方法来是如下的损失函数最小化：

$$L(f(\mathbf{x}), y) = \sum_{i=1}^N \|\boldsymbol{\beta} \mathbf{x}_i - y_i\|$$

LEMNA 使用 Fused Lasso 方法后，其目标就变成了在一定限制下使得损失函数最小化：

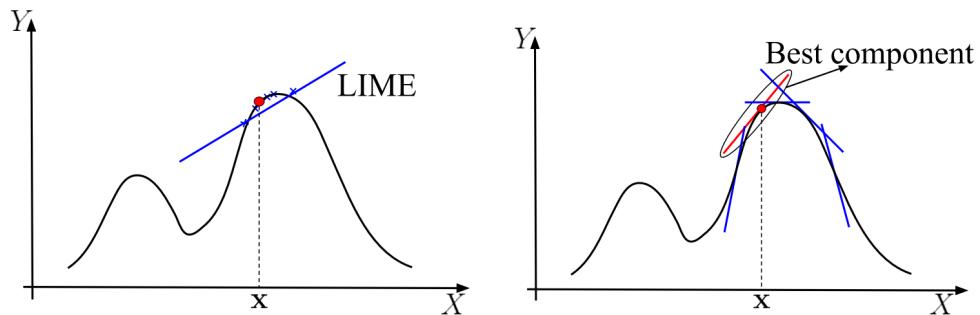
$$\begin{aligned} L(f(\mathbf{x}), y) &= \sum_{i=1}^N \|\boldsymbol{\beta} \mathbf{x}_i - y_i\|, \\ \text{subject to } &\sum_{j=2}^M \|\beta_j - \beta_{j-1}\| \leq S. \end{aligned}$$

作者也给出了，基于 Fused Lasso 方法他希望得到的最终的可解释性形式：

By introducing fused lasso in the process of approximating local decision boundary, we expect the resulting linear model to have the following form:

$$f(\mathbf{x}) = \beta_1 x_1 + \beta_2(x_2 + x_3) + \beta_3(x_4 + x_5) + \cdots + \beta_k x_M, \quad (3)$$

- Mixture Regression Model：作者的解释是，一个混合回归模型能够拟合一个局部非线性边界。换成我的话说，就是拟合一个局部分段线性模型。从作者的图里面比较容易理解：



(a) Linear regression model.      (b) Mixture regression model.

**Figure 3: Approximating a locally non-linear decision boundary. The linear regression model (a) can easily make mistakes; Our mixture regression model (b) achieves a more accurate approximation.**

可以看到，LIME 拟合的是一个局部线性模型，LEMNA 拟合的是一个分段线性模型，从示意图上可以感觉到用 分段线性模型 是更好的；下面给出这个混合回归模型的公式：

$$y = \sum_{k=1}^K \pi_k (\beta_k \mathbf{x} + \epsilon_k)$$

使用混合回归模型的目的，就是要找到上图中红色的这个线性模型，因为它对  $x$  处的拟合效果是最好的；

blue line represents an independent linear regression model. The best linear model for producing the explanation should be the red line passing through the data point  $\mathbf{x}$ . In this way, the approximation process can yield an optimal linear regression model for pinpointing important features as the explanation.

3. LEMNA 的具体实现：这部分我们结合代码一起来看会比较有感觉；

具体的代码实现，在 GitHub 上有三个相关的库：

- Henrygwb 的 [Explaining-DL](#) 库（大佬好像是跟我说这个是作者后来的工作）：这个库中的 LEMNA 实现只有 Fused Lasso 部分（我们暂时用的是这个库）；
- nitishabharathi 的 [LEMNA](#) 库：这个库中的 LEMNA 实现只有 Mixture Regression Model 部分；
- ZhihaoMeng 的 [CPSC8580](#) 库：这个库中的 LEMNA 实现很 nice，非常全，强烈建议看这个库；（虽然我还没运行过他的代码，但是我简单看了一下，实现上来说和第一个库差不多，所以花点功夫应该是可以跑起来的，有时间我也会跑一下，甚至替换到我们的项目中）

代码的实现过程中，Fused Lasso 和 Mixture Regression Model 是几乎混在一起的，下面还是简单地拆开来分析；

- Fused Lasso：调用 R 语言的实现，下面简单列一下相关的代码；

```

1 # 导入rpy的包等等操作
2 from scipy import io
3 from rpy2 import robjects
4 from rpy2.robjects.packages import importr
5 r = robjects.r
6 rpy2.robjects.numpy2ri.activate()
7 # 导入r的包
8 importr('genlasso')
9 importr('gsubfn')
10 # 使用r来实现Fused Lasso
11 X = r.matrix(data_explain)
12 Y = r.matrix(table_explain)
13 results = r.fusedlasso1d(y=Y, x=X)

```

- Mixture Regression Model：（希望看官不要嫌弃我的数学）对于原问题，最小化如下损失函数

$$L(f(\mathbf{x}), y) = \sum_{i=1}^N \|f(\mathbf{x}_i) - y_i\|,$$

$$\text{subject to } \sum_{j=2}^M \|\beta_{kj} - \beta_{k(j-1)}\| \leq S, k = 1, \dots, K$$

作者的实现依赖于“单个线性模型的预测结果的误差 满足 高斯分布”这个假设。我们将  $P(y_i | x_i)$  写成高斯混合分布的形式：

$$y_i \sim \sum_{k=1}^K \pi_k \mathcal{N}(\beta_k \mathbf{x}_i, \sigma_k^2)$$

这样就通过拟合高斯混合分布的方法来计算模型的参数，即 EM 算法；但是因为我们需要结合 Fused Lasso 方法，对 EM 算法还需要做一些修改：

- E Step：和原来的 EM 步骤一样，估计各个样本属于哪个子分布，计算期望值；
- M Step：对于参数  $\pi_k$  和  $\sigma_k$  的更新和原来的 EM 步骤是一样的，但是对于  $\beta_k$  的更新，改成使用 Fused Lasso 方法来对其进行更新；即已经用 E Step 得到了每个样本的子分布，在 M Step 中直接调用 Fused Lasso 方法对每个分布的样本进行拟合，下面给出作者的表达  
a customized procedure. That is to compute  $\beta_k$  by minimizing the following equation with respect to  $\beta_k$

$$L(x, y) = \sum_{i=1}^{N_k} \|\beta_k x_i - y_i\|, \\ \text{subject to } \sum_{j=2}^M \|\beta_{kj} - \beta_{k(j-1)}\| \leq S,$$
 (7)

where  $N_k$  refers to the number of samples assigned to the  $k^{th}$  component. Here, the reason behind this re-computation customization

这里代码就不粘了，得说一下 ZhihaoMeng 里面三个脚本的区别：

- [lemlna.py](#): (算是一个阉割版) 直接用高斯混合分布对扰动数据集 (Perturbation Samples) 进行拟合，挑选出属于最优 ( $\pi_k$  最大) 的子高斯分布的样本点，用 Fused Lasso 对这些点进行拟合；
- [lemlna-no-GMM.py](#): (算是一个精简版) 只用了 Fused Lasso 方法，这个和 Henrygwb 的库的实现是一样的；
- [lemlna-customGMM.py](#): 这是完整的 LEMNA 实现；

LEMNA 方法最后给出的特征重要性即为  $\pi_k$  最大的线性分布模型的权重  $\beta_k$ ，下面给出作者的表达：

obtain a mixture regression model enhanced by fused lasso. From this mixture model, we then identify the linear component that has the best approximation of the local decision boundary. The weights (or coefficients) in the linear model can be used to rank features. A small set of top features is selected as the explanation result.

#### 4. 我的理解：

- 为啥不用白盒梯度的方法呢：折腾来折腾去，就是拟合一个边界，感觉黑盒方法的优势并不存在；
- 筛选扰动数据集：黑盒方法的好坏很大程度上取决于你构造的扰动数据集的好坏；我们换个角度来看，LEMNA 算法最终得到的还是一个局部线性模型（混合回归模型就是多拟合几条线，我们来选一条最好的），他其实不就是用高斯混合模型对扰动数据集做了一个筛选么！
- 黑盒算法的性能受到特征维度的制约；
- 实现和理论不近相符：前面图片中我们想要得到的是红色的线性模型，即经过样本点的线性模型，但是实现过程中挑选最优的模型，则是判断混合分布的  $\pi_k$  的大小；
- 作者在实现过程中还有一个 Trick，他在 Binary 这个模型的分析过程中，设置了一个大小为 40 的分析窗口；

#### 5. 论文实验：

- 实验针对的模型：

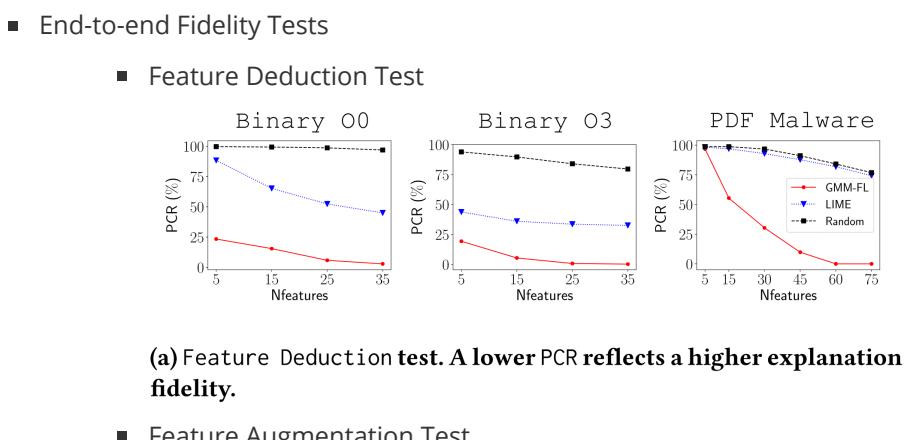
| Application | Binary Function Start |        |        |        | PDF Malware |
|-------------|-----------------------|--------|--------|--------|-------------|
|             | 00                    | 01     | 02     | 03     |             |
| Precision   | 99.99%                | 99.65% | 98.57% | 99.53% | 99.12%      |
| Recall      | 99.97%                | 99.49% | 98.81% | 99.06% | 98.13%      |
| Accuracy    | 99.99%                | 99.99% | 99.99% | 99.99% | 98.64%      |

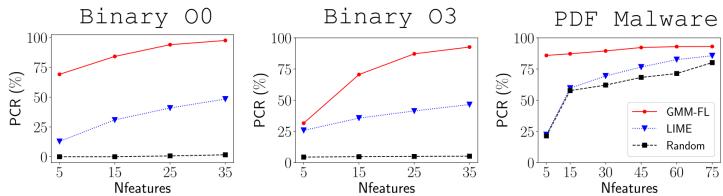
**Table 2: Classification accuracy of the trained classifiers.**

- Binary Function Start: 标记二进制代码函数起始位置的模型, 用的 RNN 模型, 其中  $O_0 \sim O_3$  表示 gcc 的编译优化等级;
- PDF Malware: PDF 恶意脚本检测的模型, 人工提取出相应的特征, 用的 MLP 模型, 因为相邻特征之间没有绝对的相关性, 所以这里没有用到 Fused Lasso;
- Baseline: 和 LIME 方法和 Random 方法进行对比;
- 如何度量精度 (Fidelity)
- Local Approximation Accuracy: 即计算线性模型预测的概率和神经网络预测的概率之间的差别, 用 RMSE 度量, 公式如下
$$RMSE = \sqrt{\frac{\sum_{i=1}^n (p_i - \hat{p}_i)^2}{n}}$$
- End-to-end Fidelity Tests: 主要思想是保留、删除、生成重要特征, 观察模型结果的变化, 都用 PCR 来度量
  - Feature Deduction Test: 在原始数据中删除重要特征, PCR 越小说明解释方法越好;
  - Feature Augmentation Test: 挑选一个相反类 (opposite class) 数据, 在数据中增加 (从原始数据中分析得到的) 重要特征, PCR 越大说明解释方法越好;
  - Synthetic Test: 在空白数据中只添加 (从原始数据中分析得到的) 重要特征, PRC 越大说明解释方法越好;
- 实验结果:
  - Local Approximation Accuracy: LEMNA 方法的 RMSE 结果优于 LIME;

| Method | Binary Function Start |        |        |        | PDF malware |
|--------|-----------------------|--------|--------|--------|-------------|
|        | 00                    | 01     | 02     | 03     |             |
| LIME   | 0.1784                | 0.1532 | 0.1527 | 0.1750 | 0.1178      |
| LEMNA  | 0.0102                | 0.0196 | 0.0113 | 0.0110 | 0.0264      |

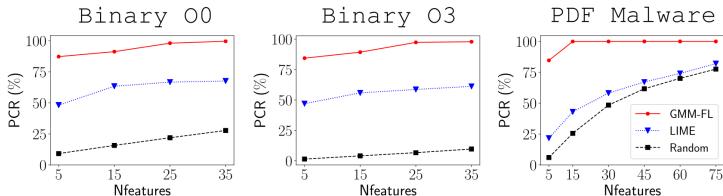
**Table 3: The Root Mean Square Error (RMSE) of local approximation. LEMNA is more accurate than LIME.**





(b) Feature Augmentation test. A higher PCR reflects a higher explanation fidelity.

#### ■ Synthetic Test



(c) Synthetic test. A higher PCR reflects a higher explanation fidelity.

- 分析：LEMNA 方法均优于 LIME，并且发现对于 PDF Malware 这个网络，LIME 方法的效果和 Random 差不多；

Across all three tests, our LEMNA outperforms LIME and the random baseline by a big margin. Interestingly, for the malware classifier, LIME performs as poor as random feature selection. This is because the feature vectors are sparse, which hurts the “smoothness” of the decision boundary. LIME has a hard time to accurately approximate the non-smooth boundary, which again validates our design intuition. Our system is more suitable for security applications, considering that security applications require a much higher explanation precision compared to image analysis tasks.

- 如何利用可解释方法：作者最后给出了几种有意思的可解决方法的利用场景，有兴趣可以再看看；

## Links

- 论文链接：[Guo W, Mu D, Xu J, et al. Lemma: Explaining deep learning based security applications\[C\]//proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018: 364-379.](#)
- 代码链接：[Black-box explanation of deep learning models using mixture regression models](#)
- 代码链接：<https://github.com/nitishshabharathi/LEMNA>
- 代码链接：<https://github.com/ZhihaoMeng/CPSC8580>

## Robust Classification with Convolutional Prototype Learning

### Contribution

- 提出了基于原型学习的图像分类网络，是[文章](#)的前身；

## Links

- 论文链接: [Yang H M, Zhang X Y, Yin F, et al. Robust classification with convolutional prototype learning\[C\]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 3474-3482.](#)

# Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions

## Contribution

- 通过原型 (Prototypes) 学习, 来实现可自解释的模型;

## Notes

### 1. 文章方法

首先看一下整个 **模型的框架图**: 可以看到这个模型可以分为编码器 (用来把图像编码到低维超平面)、解码器 (用来把低维超平面向量解码为图像) 和原型分类网络 (在低维超平面上进行自解释和分类任务)

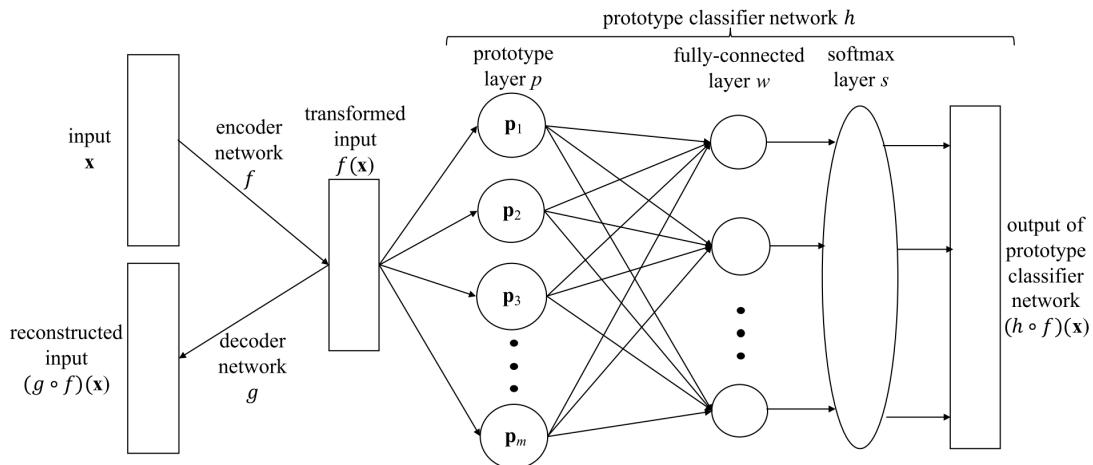


Figure 1: Network Architecture

**编码器**  $f : \mathbb{R}^p \rightarrow \mathbb{R}^q$ , 把高维的图片向量编码为低维的超平面向量, 主要是为了减少原型分类网络的输入维度, 利于网络收敛; (这里是否应该讨论一下不同的低维空间对最后结果的影响?)

**解码器**  $g : \mathbb{R}^q \rightarrow \mathbb{R}^p$ , 把低维的超平面向量解码为高维的图片向量, 主要是为了可以可视化原型分类网络学到的各个原型;

**原型分类网络**  $h : \mathbb{R}^q \rightarrow \mathbb{R}^K$ , 把低维的超平面向量分类为目标的  $K$  类; 这里比较特殊的是其中的原型层 (Prototypes Layer), 原型层中包含  $m$  个原型向量  $p_1, p_2, \dots, p_m \in \mathbb{R}^q$ , 模型学习完后通过可视化层后可以看到每个原型都可能学习到了其中一类分类的特征, 原型层的输出是一个  $L_2$  距离向量, 具体公式如下:

$$p(\mathbf{z}) = [\|\mathbf{z} - \mathbf{p}_1\|_2^2, \|\mathbf{z} - \mathbf{p}_2\|_2^2, \dots, \|\mathbf{z} - \mathbf{p}_m\|_2^2]^T$$

**损失函数** 的设计:

- 保证模型分类的准确性, 使用交叉熵损失函数:

$$E(h \circ f, D) = \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K -\mathbb{1}[y_i = k] \log((h \circ f)_k(\mathbf{x}_i))$$

- 保证编码器解码器的重构误差，使用  $L_2$  距离损失函数：

$$R(g \circ f, D) = \frac{1}{n} \sum_{i=1}^n \|(g \circ f)(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$$

- 保证模型的自解释性，使用两个  $L_2$  正则化项：

$$R_1(\mathbf{p}_1, \dots, \mathbf{p}_m, D) = \frac{1}{m} \sum_{j=1}^m \min_{i \in [1, n]} \|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2,$$

$$R_2(\mathbf{p}_1, \dots, \mathbf{p}_m, D) = \frac{1}{n} \sum_{i=1}^n \min_{j \in [1, m]} \|f(\mathbf{x}_i) - \mathbf{p}_j\|_2^2.$$

其中，第一项用来 保证每个输入都能和至少一个原型对应；第二项用来 保证每个原型都能和至少一个输入对应。

- 整体损失函数如下：

$$\begin{aligned} L((f, g, h), D) &= E(h \circ f, D) + \lambda R(g \circ f, D) \\ &\quad + \lambda_1 R_1(\mathbf{p}_1, \dots, \mathbf{p}_m, D) \\ &\quad + \lambda_2 R_2(\mathbf{p}_1, \dots, \mathbf{p}_m, D), \end{aligned}$$

## 2. 实验：Handwritten Digits

- 原任务的精度：在 MNIST 测试集上的精度为 99.22；（作者直接用了 15 个原型进行训练，并没有讨论原型个数对最后结果的影响）
- 可视化学习到的原型：可以看到和真实的手写数字非常的相近；



## Links

- 论文链接：[Li O, Liu H, Chen C, et al. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions\[C\]//Proceedings of the AAAI Conference on Artificial Intelligence. 2018, 32\(1\).](#)
- 论文代码：[PrototypeDNN](#)

# Notes

这篇是纪老师关于模型可解释性文章的综述，以下均为个人对文章做的笔记，建议自己阅读原文。另外纪老师在这方面有很多不错的工作，值得关注。

模型可解释性方向有特别多的工作，文章中也有一些不全的地方需要自己去思考和补充。

## 机器学习可解释性问题

1. **模型可解释性问题**: 可解释性旨在帮助人们理解机器学习模型是如何学习的，它从数据中学到了什么，针对每一个输入它为什么做出如此决策以及它所做的决策是否可靠；
2. **模型的复杂度与模型准确性相关联，又与模型的可解释性相对立。**
3. 根据选择结构简单易于解释的模型然后训练它，还是训练复杂的最优模型然后开发可解释性技术解释它，将机器学习模型可解释性总体上分为：**ante-hoc 可解释性和 post-hoc 可解释性**；

### \* Ante-hoc 可解释性

1. Ante-hoc 可解释性指**模型本身内置可解释性**，即对于一个已训练好的学习模型，我们无需额外的信息就可以理解模型的决策过程或决策依据；
2. 自解释模型：
  - (1) 可模拟性：在一定时间可以预测模型的每一步计算；
  - (2) 可分解性：模型的每个部分都可以得到一个直观的解释；
  - (3) 结构简单：由于人类认知的局限性，自解释模型的内置可解释性受模型的复杂度制约，这要求自解释模型结构一定不能过于复杂；
3. 广义加性模型：在简单模型和复杂问题之间的一个折中；  
$$g(y) = f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$$
4. 注意力机制：通过 Attention 的权重来分析模型关注的重点是什么；

### Post-hoc 可解释性

1. Post-hoc 可解释性的重点在于设计高保真的解释方法或构建高精度的解释模型，根据解释目的和解释对象的不同，可分为全局可解释性和局部可解释性；
2. 经典的解释方法如下：

Table 1 Summary of classic post-hoc interpretation methods.

表 1 经典的 post-hoc 解释方法总结

| Method                 | G/L  | MA/MS | TML | FCN | CNN | RNN | Fidelity | Security | Domain      |
|------------------------|------|-------|-----|-----|-----|-----|----------|----------|-------------|
| inTree [23]            | G    | MS    | ✓   | ✗   | ✗   | ✗   | ○        | -        | n/a         |
| SGL [47]               | G    | MS    | ✓   | ✗   | ✗   | ✗   | ○        | -        | n/a         |
| GIRP [53]              | G    | MA    | ✓   | ✓   | ✓   | ✓   | ○        | ✗        | CV/NLP      |
| MAGIX [58]             | G    | MA    | ✓   | ✓   | ✓   | ✓   | ○        | -        | n/a         |
| DeepVID [70]           | G    | MA    | ✗   | ✗   | ✓   | ✗   | ○        | ✗        | CV          |
| AM [75]                | G    | MS    | ✗   | ✓   | ✓   | ✗   | ●        | ✗        | CV          |
| Nguyen et al. [79]     | G    | MS    | ✗   | ✓   | ✓   | ✗   | ●        | ✗        | CV          |
| Yuan et al. [82]       | G    | MS    | ✗   | ✗   | ✗   | ✓   | ●        | ✗        | NLP         |
| Saliency Mask [93]     | L    | MA    | ✗   | ✓   | ✓   | ✗   | ○        | ✗        | CV          |
| RSRS [94]              | L    | MA    | ✗   | ✓   | ✓   | ✗   | ○        | ✗        | CV          |
| LIME [13]              | L    | MA    | ✓   | ✓   | ✓   | ✓   | ●        | ✗        | CV/NLP      |
| LORE [96]              | L    | MA    | ✓   | ✓   | ✓   | ✓   | ○        | ✗        | n/a         |
| Anchor [98]            | L    | MA    | ✓   | ✓   | ✓   | ✓   | ●        | ✗        | CV/NLP      |
| LEMNA [99]             | L    | MS    | ✗   | ✗   | ✗   | ✓   | ●        | ✗        | NLP/Malware |
| Grad [73]              | L    | MS    | ✗   | ✓   | ✓   | ✓   | ○        | ✗        | CV/NLP      |
| DeconvNet [80]         | L    | MS    | ✗   | ✗   | ✓   | ✗   | ●        | ✗        | CV          |
| GuidedBP [100]         | L    | MS    | ✗   | ✗   | ✓   | ✗   | ●        | ✗        | CV          |
| Integrated [101]       | L    | MS    | ✗   | ✓   | ✓   | ✓   | ○        | ✗        | CV/NLP      |
| SmoothGrad [102]       | L    | MS    | ✗   | ✓   | ✓   | ✓   | ●        | ✗        | CV/NLP      |
| LRP [105]              | L    | MS    | ✗   | ✓   | ✓   | ✓   | ●        | ✗        | CV/NLP      |
| DeepLIFT [106]         | L    | MS    | ✗   | ✓   | ✓   | ✓   | ●        | ✗        | CV/Genomics |
| Guided Inversion [103] | L    | MS    | ✗   | ✗   | ✓   | ✗   | ●        | ✓        | CV          |
| CAM [112]              | L    | MS    | ✗   | ✗   | ✓   | ✗   | ●        | ✗        | CV          |
| Grad-CAM [113]         | L    | MS    | ✗   | ✗   | ✓   | ✗   | ●        | ✗        | CV          |
| AI <sup>2</sup> [115]  | L    | MS    | ✗   | ✗   | ✓   | ✗   | ○        | ✓        | CV          |
| OpenBox [116]          | G, L | MS    | ✗   | ✓   | ✗   | ✗   | ●        | ✓        | CV          |

Note: G = global, L = local, MA = model-agnostic, MS = model-specific, TML = traditional machine learning, ○ = low, ● = middle, ● = high, - = unknown, CV = computer vision, NLP = natural language processing, and n/a = not mentioned in the literature.

3. 全局解释：全局可解释性旨在帮助人们从整体上理解模型背后的复杂逻辑以及内部的工作机制，如模型是如何学习的、模型从训练数据中学到了什么、模型是如何进行决策的等；
- (1) \* 规则提取：通过受训模型中提取解释规则的方式，提供对复杂模型尤其是黑盒模型整体决策逻辑的理解（比较早期的做法）；
  - (2) 模型蒸馏：
    - 定义：通过降低模型复杂度，解决理解受训模型比较困难的问题，是一种经典的模型压缩方法；
    - 核心思想：利用结构紧凑的学生模型来模拟结构复杂的教师模型，从而完成从教师模型到学生模型的知识迁移过程，实现对复杂教师模型的知识的“蒸馏”；
    - 训练损失函数定义：  

$$L_{student} = \alpha L^{(soft)} + (1 - \alpha)L^{(hard)}$$
 其中， $L^{(soft)}$  为软目标损失，期望学生模型能够学到教师模型相似的概率分布输出；  
 $L^{(hard)}$  为硬目标损失，要求学生模型能够保留教师模型决策的类别；
    - 模型蒸馏解释方法实现简单，易于理解，且不依赖待解释模型的具体结构信息，因而作为一种模型无关的解释方法，常被用于解释黑盒机器学习模型；
    - 蒸馏模型只是对原始复杂模型的一种全局近似，基于蒸馏模型所做出的解释不一定能够反映待解释模型的真实形为；

我的想法：1. 能否通过模型蒸馏的方法去生成一个黑盒模型的替代模型，从而去辅助生成对抗样本？（这个关键在于生成的替代模型能否较好地逼近黑盒模型）2. 模型蒸馏的方法，最终能够得到怎样的模型解释？能否通过模型蒸馏的方法来解释现有的语音识别模型？（这个关键在于如何去分析蒸馏以后的模型）

### (3) 激活最大化 (Activation Maximization):

- 定义：通过在特定的层上**找到神经元的首选输入来最大化神经元激活**，来理解 DNN 中每一层隐含层神经元所捕获的表征；
- 核心思想：通过寻找有界范数的输入模式，最大限度地激活给定地隐藏神经元，而一个单元最大限度地响应的输入模式可能是“一个单元正在做什么的”良好的一阶表示；
- 形式化定义：

$$x^* = \arg \max_x f_l(x) - \lambda \|x\|^2$$

其中左边项期望  $x$  能够使得当前神经元的激活值最大；右边项期望  $x$  与原样本尽可能接近（右边应该改用  $\Delta x$  来表示）；

- 激活最大化解释方法是一种模型相关的解释方法，相比规则提取解释和模型蒸馏解释，其解释结果更准确，更能反映待解释模型的真实形为；
- 激活最大化本身是一个优化问题，在通过激活最大化寻找原型样本的过程中，优化过程中的噪音和不确定性可能导致产生的原型样本难以解释；
- 激活最大化解释方法**难以用于解释自然语言处理模型和图神经网络模型**；

## 4. 局部解释：模型的局部可解释性以输入样本为导向，通常可以通过分析输入样本的每一维特征对模型最终决策结果的贡献来实现。

### (1) 敏感性分析 (Sensitivity Analysis):

- 核心思想：通过**逐一改变自变量的值来解释因变量受自变量变化影响大小的规律**；
- 根据**是否需要利用模型的梯度信息**，敏感性分析方法可分为模型相关方法和模型无关方法；
- **模型相关方法**：利用模型的局部梯度信息评估特征与决策结果的相关性，常见的相关性定义如下：

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

即为模型梯度的  $l_2$  范数分解；

- **模型无关方法**：无需利用模型的梯度信息，只关注**待解释样本特征值变化对模型最终决策结果的影响**。具体地，该方法通过观察去掉某一特定属性前后模型预测结果的变化来确定该属性对预测结果的重要性，即：

$$R_i(x) = f(x) - f(x \setminus x_i)$$

- 敏感性分析方法提供的解释结果通常**相对粗糙且难以理解**，且无法分析**多个特征之间的相关关系**；

### (2) 局部近似：

- 核心思想：利用**结构简单的可解释模型拟合待解释模型针对某一输入实例的决策结果**，然后基于解释模型对该决策结果进行解释；
- 基于局部近似的解释方法实现简单，易于理解且不依赖待解释模型的具体结构，适于**解释黑盒机器学习模型**；
- 对于每个输入样本都需要训练一个解释模型，**效率不高**，并且该方法基于**特征相互独立的假设**；

### (3) ☆ 反向传播：

- 核心思想：利用**DNN 的反向传播机制**将模型的决策重要性信号从模型的输出层神经元逐层传播到模型的输入以推导输入样本的特征重要性；

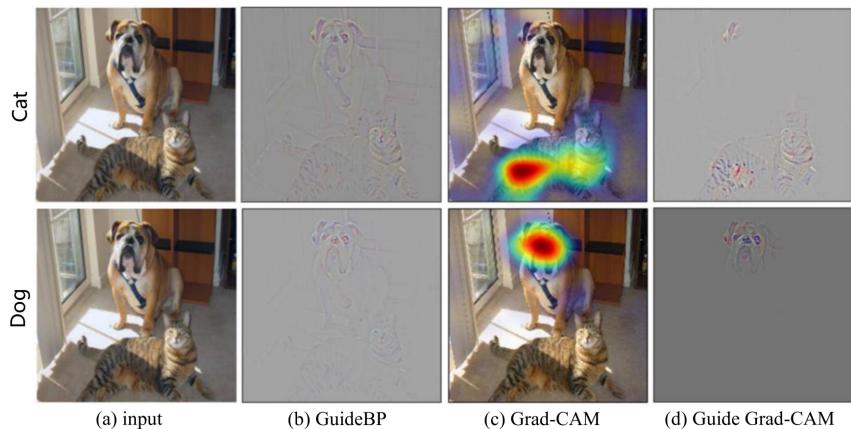
- 其中，**SmoothGrad 方法**的核心思想：通过向待解释样本中添加噪声对相似的样本进行采样，然后利用反向传播方法求解每个采样样本的决策显著图，最后将所有求解得到的显著图进行平均并将其作为对模型针对该样本的决策结果的解释；
- **分层相关性传播(LRP)方法**的核心思想：利用反向传播将高层的相关性分值递归地传播到底层直至传播到输入层；
- 基于反向传播地解释方法通常实现简单、计算效率高且充分利用了模型的结构特性；
- 如果预测函数在输入附近变得平坦，那么预测函数相对于输入的梯度在该输入附近将变得很小，进而导致无法利用梯度信息定位样本的决策特征；

反向传播的想法应该可以在语音领域进行尝试。

#### (4) \* 特征反演 (Feature Inversion):

- 定义：特征反演作为一种可视化和理解 DNN 中间特征表征的技术，可以充分利用模型的中间层信息，以提供对模型整体行为及模型决策结果的解释；
- 特征反演解释方法分为：模型级解释方法和实例级解释方法；

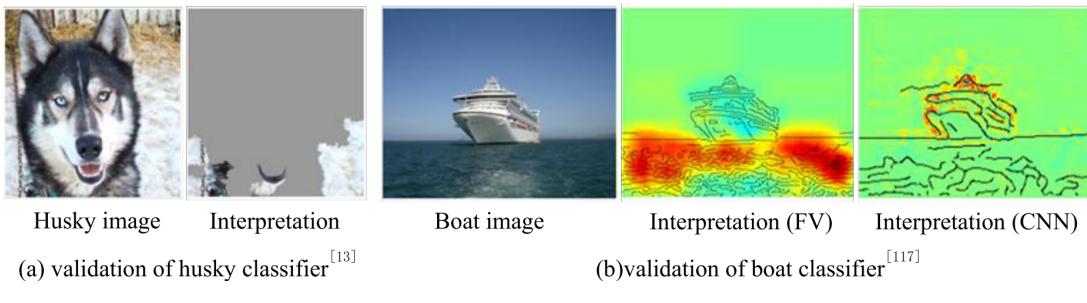
#### (5) 类激活映射：



- ☆ 前提：**CNN 不同层次的卷积单元包含大量的位置信息，使其具有良好的定位能力。**然而，传统 CNN 模型通常在卷积和池化之后采用全连接层对卷积层提取的特征图进行组合用于最终的决策，因而**导致网络的定位能力丧失**，这个问题需要在这些工作中解决。
- **类激活映射 (Class Activation Mapping, CAM) 解释方法**：利用全局平均池化(Global Average Pooling)层来替代传统 CNN 模型中除 softmax 层以外的所有全连接层，并通过将输出层的权重投影到卷积特征图来识别图像中的重要区域 => (**需要用全局平均池化层替换模型中的全连接层并重新训练**)。
- **梯度加权类激活映射 (Grad-CAM) 解释方法**：给定一个输入样本，Grad-CAM 首先计算目标类别相对于最后一个卷积层中每一个特征图的梯度并对梯度进行全局平均池化，以获得每个特征图的重要性权重；然后，基于重要性权重计算特征图的加权激活，以获得一个粗粒度的梯度加权类激活图，用于定位输入样本中具有类判别性的重要区域 => (**不需要修改网络后进行重训练**)；
- **导向梯度加权类激活 (Guided Grad-CAM) 解释方法**：即将 GuidedBP 方法和 Grad-CAM 方法进行结合；

## 可解释性应用

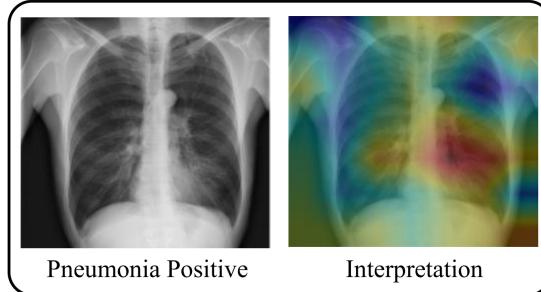
1. 模型验证：由于数据集中可能存在偏差，并且验证集也可能与训练集同分布，我们很难简单地通过评估模型在验证集上的泛化能力来验证模型的可靠性，也很难验证模型是否从训练数据中学到了真正的决策只是。这里通过可解释性方法，去分析模型在做决策时到底更加关注哪些特征，这些特征是否是合理的。如下：



2. \* 模型诊断：诊断模型中的缺陷；

这个东西感觉不太靠谱，模型可解释性的方法很大程度上并不能得到很直观的一个解释，又如何依靠模型可解释性来做模型的诊断。

3. \* 辅助分析：如医疗领域，使用可解释性方法来辅助医护人员进行检查；



4. 知识发现：辅助人学习基于大量数据训练的模型中的知识；

- LEMNA 解释方法可以挖掘出检测模型从数据中学到的新知识；

## 可解释性与安全性分析

应该仔细思考：**可解释性方法和安全领域的相关关系，从而去发掘新的应用场景、攻击场景、防御手段等。**

1. 安全隐患消除：模型可解释性方法用于检测对抗样本，并增强模型的鲁棒性；
2. 安全隐患：**攻击者也同样可以利用模型可解释性的方法来生成更好的对抗样本；**
3. ☆ 自身安全问题：由于采用了近似处理或是基于优化手段，大多数解释方法只能提供近似的解释，因而解释结果与模型的真是形为之间存在一定的不一致性
  - 不改变模型的决策结果的前提下，使解释方法解释出错：
$$\arg \max_{\delta} D(I(x_t; N), I(x_t + \delta; N)) s.t. \|\delta\|_{\infty} \leq \epsilon, f(x_t + \delta) = f(x_t)$$

其中， $I(x_t; N)$  为解释系统对神经网络  $N$  针对样本  $x_t$  决策结果  $f(x_t)$  的解释。

  - 模型结果出错，单不改变解释方法解释出错；

## 未来方向

1. ☆如何设计更精确、更友好的解释方法，消除解释结果与模型真实形为之间的不一致；
2. ☆如何设计更科学、更统一的可解释性评估指标，以评估可解释方法解释性能和安全性；

探讨一下：（我）老板问的问题是：“模型可解释性这个东西做出来，你希望它能有怎样的效果？我们要用这个要求去检验模型可解释性”；老板第二句话是：“我们应该是去做一些开创性的东西，而不是说，在别人的方法上面修修补补，或者是迁移到另一个领域，然后发一篇文章就了事了”。最后一句：“这个评估是这个领域亟待解决的问题，你能不能提出一个通用、合理的方法来评估这样方法，如果不行的话，针对不同的分类，我们能不能来做评估”。

## Links

- 论文链接：[纪守领, et al. "机器学习模型可解释性方法, 应用与安全研究综述." 计算机研究与发展 56.10 \(2019\).](#)

# Regional Tree Regularization for Interpretability in Black Box Models

---

## Contribution

- 文章的延续工作；
- 是一个 Regional Explanation 方法：对每一个小的区域拟合一颗决策树。然后用决策树的复杂度作为惩罚项来优化网络的训练；

## Notes

我觉得这种方法就挺离谱的，也没什么笔记值得记得，问题一大堆，就记一些我能直接想到地：

- 两个方法的思想是：用决策树来拟合目标模型，然后用决策树的复杂度作为惩罚项来训练网络，希望网络的决策尽可能地简单。从这个思想上面，最后还是要用决策树来对网络进行解释的，那这个和我直接训练一个决策树有什么区别？这个可解释性方法有什么意义？
- 对整个网络拟合一个决策树，这样够嘛？对于复杂数据集，就用一个决策树，人能理解嘛？
- 为了拟合这个决策树，你怎么来采样数据呢？通过全部数据显然是不可能的啊，你怎么保证采样是合理的？
- 拟合了决策树以后，确实拿到了“采样的数据集”在树上的复杂度，结果作者因为它不可导，再去训练一个MLP，什么意思？嫌我方法还不够黑盒？
- 算法的速度怎么来保证？我要训练决策树，我要训练MLP？我还要训练我的黑盒网络？训练完了精度还不一定行，我再重新训练，世界末日了？
- 对于复杂的数据集，方法二怎么来定义“Region”，定义了“Region”以采样的问题怎么来解决？
- 文章总得来说，是想实现全局的自解释模型，但是我觉得它的思路错了；

## Links

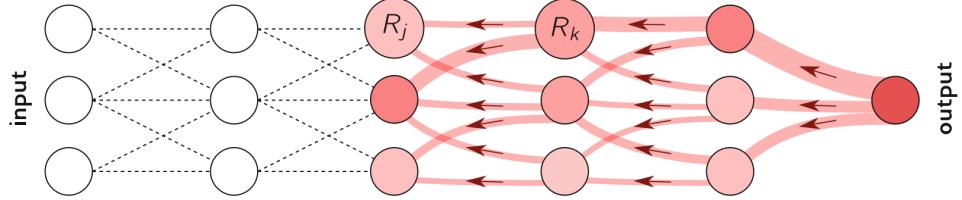
- 论文链接：[Wu M, Parbhoo S, Hughes M, et al. Regional tree regularization for interpretability in black box models\[J\]. arXiv preprint arXiv:1908.04494, 2019.](#)

# Layer-Wise Relevance Propagation: An Overview

---

## Notes

- ☆**信息守恒**：LRP 实现的传播过程服从守恒性质，神经元接收到的信息必须等量地重新分配到下一层。简单来看，LRP 就是一种权重不断回传的算法，如下图所示



**Fig. 10.2.** Illustration of the LRP procedure. Each neuron redistributes to the lower layer as much as it has received from the higher layer.

## 2. LRP Rules for Deep Rectifier Networks

该部分主要针对 ReLU 神经网络，为了一般化，文章设  $a_0 = 1$  并且  $w_{0k}$  是偏置项：

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

- **Basic Rule (LRP-0):** 即直接按照比例进行回传；

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

作者指出：尽管这条规则看起来很直观，但可以证明，这条规则应用在整个神经网络中时，等效于  $\text{Grad} \times \text{Input}$ ；

- **Epsilon Rule (LRP- $\epsilon$ ):** 即在相关性中引入一个  $\epsilon$ ；

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

作者指出：添加  $\epsilon$  项可以用来稀释贡献较弱 (weak) 或相反 (contradictory) 的项，当其值变大时，只有那些影响大的项的结果才会被保留，这可以用来产生更稀疏，噪声更小的相关性解释；

- **Gamma Rule (LRP- $\gamma$ ):** 即在相关性中引入一个正向系数的比例因子  $\gamma$ ；

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

- 算法实现：将上述三种规则统一为如下形式

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

这里， $\rho$  主要和  $\gamma$  相关。上述公式在实际中可以分成 4 步进行实现

---


$$\forall_k : z_k = \epsilon + \sum_{0,j} a_j \cdot \rho(w_{jk}) \quad (\text{forward pass})$$

$$\forall_k : s_k = R_k / z_k \quad (\text{element-wise division})$$

$$\forall_j : c_j = \sum_k \rho(w_{jk}) \cdot s_k \quad (\text{backward pass})$$

$$\forall_j : R_j = a_j c_j \quad (\text{element-wise product})$$


---

其中第三步可以借助深度学习框架中的梯度下降算法来实现

$$c_j = [\nabla \left( \sum_k z_k(\mathbf{a}) \cdot s_k \right)]_j$$

作者给出了可行的 **PyTorch** 实现

---

```

def relprop(a,layer,R):
    z = epsilon + rho(layer).forward(a)
    s = R/(z+1e-9)
    (z*s.data).sum().backward()
    c = a.grad
    R = a*c

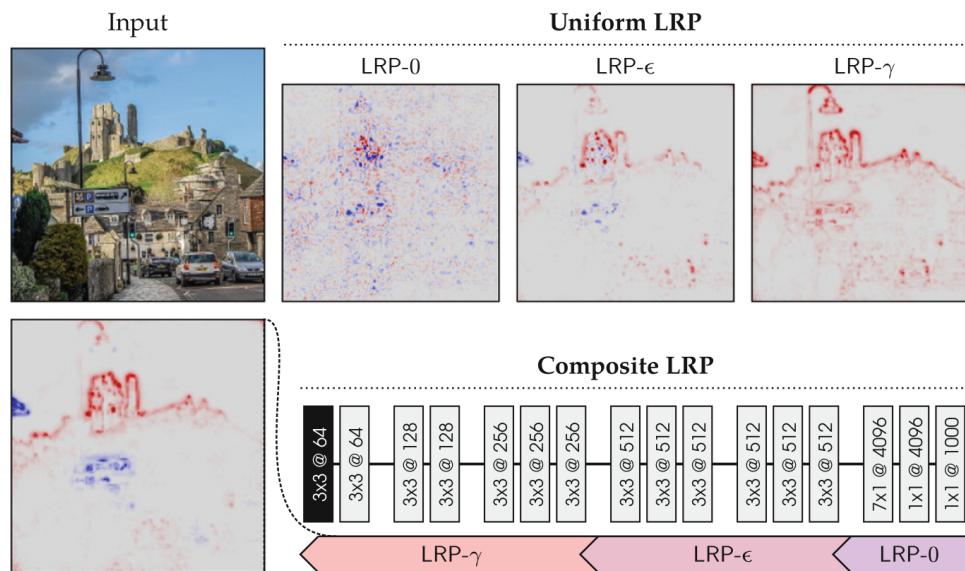
return R

```

---

### 3. ☆ 不同的神经网络层选择不同的 LRP 策略

- 不同的 LRP 策略给解释结果带来的影响，如下图：

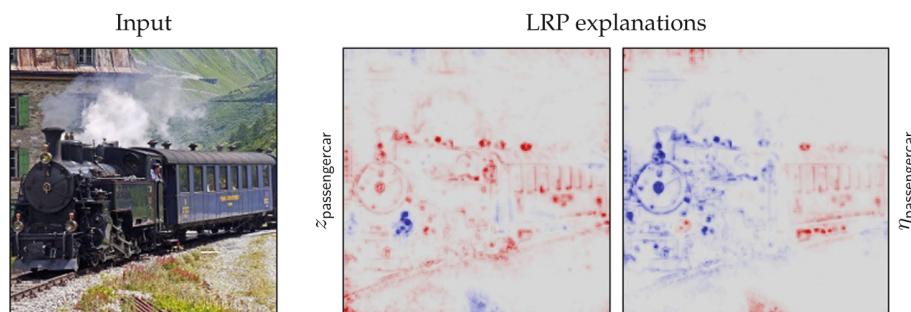


**Fig. 10.4.** Input image and pixel-wise explanations of the output neuron ‘castle’ obtained with various LRP procedures. Parameters are  $\epsilon = 0.25$  std and  $\gamma = 0.25$ .

- Softmax-Layer：神经网络的最后一层常见的都是使用 Softmax 输出模型最后的概率，公式如下

$$\text{affine: } z_c = \sum_{0,k} a_k w_{kc} \text{softmax: } P(\omega_c) = \frac{\exp(z_c)}{\sum_{c'} \exp z_{c'}}$$

如下图中间图所示



**Fig. 10.5.** Explanations obtained for the output neuron ‘passenger\_car’ and for the actual probability of the class ‘passenger\_car’. The locomotive switches from positive to negatively relevant.

如果直接对  $z_c$  进行解释，可以发现模型结果与“乘用车”部分有较大的相关性，但是对于“火车头”仍然有较大的相关性，所以这样的可解释结果是不准确的（如何区分模型的可解释性结果是模型导致的问题，还是解释性方法导致的问题？）；

作者指出，可以转换成对  $\eta_c$  的解释，效果如上图右边图所示，可以发现这次的模型解释结果更多得是在关注“乘用车”，而“火车头”部分则是出现了负相关的情况。公式如下：

$$\eta_c = \log^{P(\omega_c)/(1-P(\omega_c))}$$

该公式可以通过两层计算得到：（具体的实现如何做？）

$$z_{c,c'} = \sum_{0,k} a_k (w_{kc} - w_{kc'})$$

$$\eta_c = -\log \sum_{c' \neq c} \exp(-z_{c,c'})$$

第一层公式用来计算 the log-probability ratios -  $\log^{P(\omega_c)/P(\omega_{c'})}$ ，第二层公式用来计算 log-sum-exp pooling。对于这种池化操作，可以通过如下公式进行相关性系数的回传：

$$R_{c,c'} = z_{c,c'} \cdot \exp(-z_{c,c'}) / \sum_{c'' \neq c} \exp(-z_{c,c''})$$

- Special Pooling Layers：在卷积网络中经常用到一些池化层
  - Sum Pooling：可以采用正常的 linear-ReLU 层对这层进行替换；
  - Max Pooling：可以采用 winner-take-all 策略，或是和 Sum Pooling 采用相同的策略；
- Batch Normalization Layers：批量池化层在固定后起到的作用只是对输入起到了修改均值和缩放的作用，所以可以借助想用的线性仿射变换进行替代；
- Input Layer：对于像素点的输入层，作者指出可以使用  $z^B$ -rule，具体公式如下

$$R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j$$

- LSTM Layer：其公式如下，作者指出一种常用的实现是将相关性系数沿着信号传递（即只沿着第二项进行传递）

$$h_k = \text{sigm}(\sum_j a_j v_{jk} + c_k) \cdot g(\sum_j a_j w_{jk} + b_k)$$

- Pairwise Matching：常用在推荐系统或者图像-特征图匹配中（这个场景太模糊了，有没有更具体的解释？），作者指出一种可行的实现是构造如下神经元

$$a_k = \max(0, \sum_i x_i w_{ik}) \cdot \max(0, \sum_j y_j v_{jk})$$

其相关性的传递公式如下

$$R_{ij} = \sum_k \frac{x_i y_j w_{ik} v_{jk}}{\sum_{ij} x_i y_j w_{ik} v_{jk}} R_k$$

- 常用的实现列表：

| Name                   | Formula  | Usage                          | DTD        |
|------------------------|--|--------------------------------|------------|
| LRP-0 [7]              | $R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$  | Upper layers                   | ✓          |
| LRP- $\epsilon$ [7]    | $R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k$   | Middle layers                  | ✓          |
| LRP- $\gamma$          | $R_j = \sum_k \frac{a_j (w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j (w_{jk} + \gamma w_{jk}^+)} R_k$  | Lower layers                   | ✓          |
| LRP- $\alpha\beta$ [7] | $R_j = \sum_k \left( \alpha \frac{(a_j w_{jk})^+}{\sum_{0,j} (a_j w_{jk})^+} - \beta \frac{(a_j w_{jk})^-}{\sum_{0,j} (a_j w_{jk})^-} \right) R_k$ | Lower layers                   | $\times^a$ |
| flat [30]              | $R_j = \sum_k \frac{1}{\sum_j 1} R_k$  | Lower layers                   | $\times$   |
| $w^2$ -rule [36]       | $R_i = \sum_j \frac{w_{ij}^2}{\sum_i w_{ij}^2} R_j$  | First layer ( $\mathbb{R}^d$ ) | ✓          |
| $z^B$ -rule [36]       | $R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j$                                | First layer (pixels)           | ✓          |

(<sup>a</sup>DTD interpretation only for the case  $\alpha = 1, \beta = 0.$ )

## Links

- 论文链接: [Montavon G, Binder A, Lapuschkin S, et al. Layer-wise relevance propagation: an overview\[J\]. Explainable AI: interpreting, explaining and visualizing deep learning, 2019: 193-209.](#)
- 论文代码: <https://git.tu-berlin.de/gmontavon/lrp-tutorial>