KSME 인공지능머신연구회 고급인공지능 강습회, January 19-21, 2022

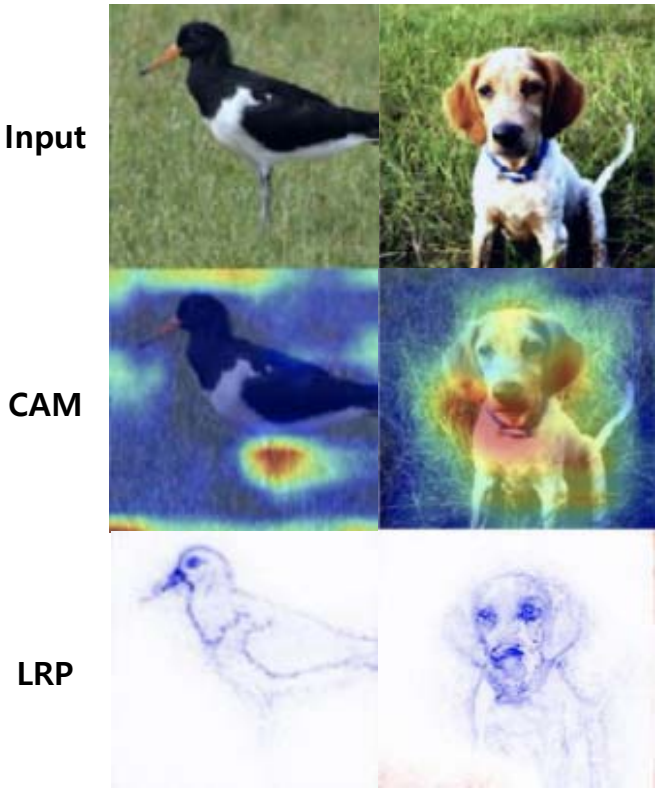
Layer-wise Relevance Propagation

Prof. Hyunseok Oh

School of Mechanical Engineering
Gwangju Institute of Science and Technology

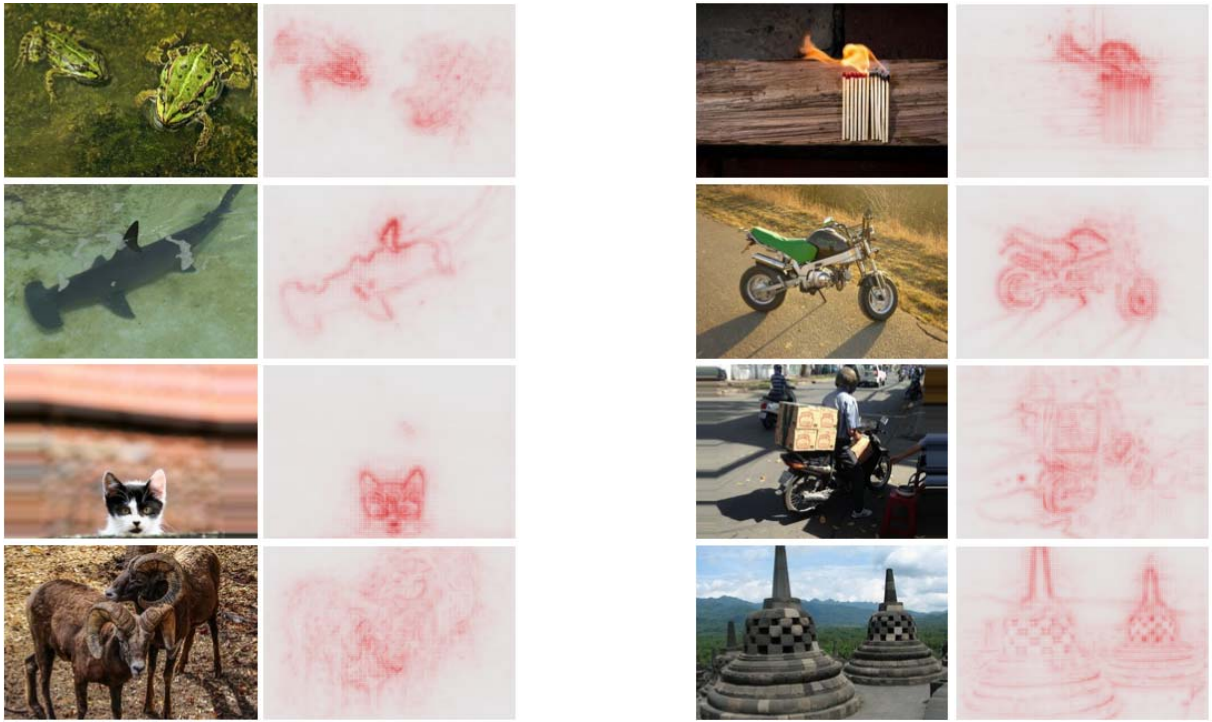
CAM vs LRP

- CAM (and Grad-CAM)
  - 기울기 값을 활용하여 커널의 가중치를 표현
  - 결과물이 등고선 모양(Heat map)으로 나타남
- LRP
  - 출력값의 분류 정보가 픽셀 단위 값으로 분해
  - 각 뉴런의 출력값을 '기여도'로 표현함
  - 결과물이 점 구름과 같이 나타남



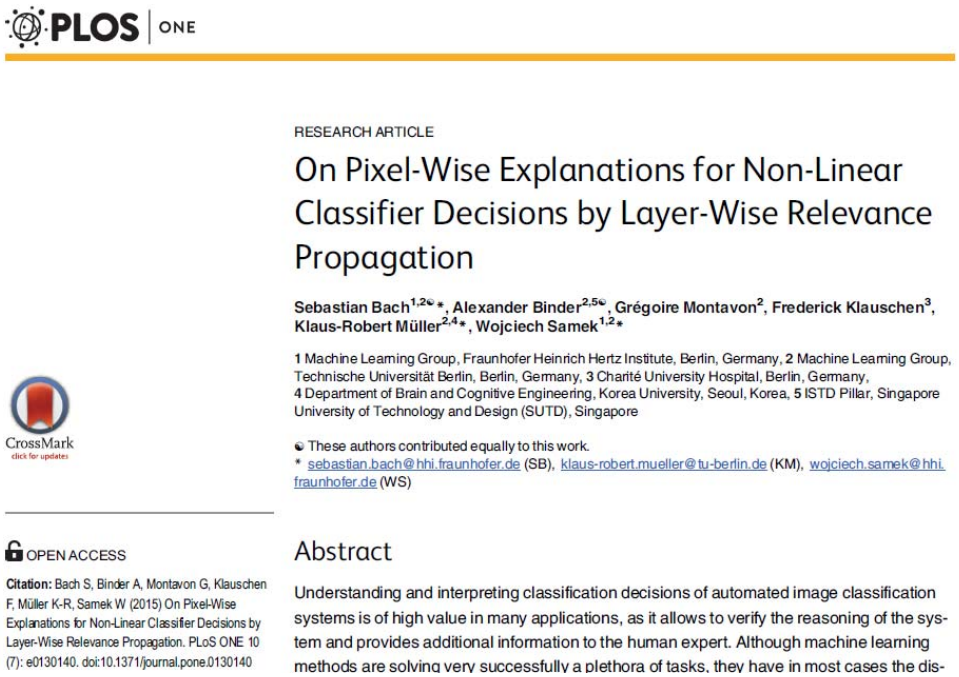
LRP Example

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - GoogleNet을 기반으로한 LRP 예시



Layer-wise Relevance Propagation (LRP)

- Bach et al., “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation”, PLOS ONE, 2015.



PLOS | ONE

RESEARCH ARTICLE

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

Sebastian Bach[1,2☯*], Alexander Binder[2,5☯], Grégoire Montavon[2], Frederick Klauschen[3], Klaus-Robert Müller[2,4*], Wojciech Samek[1,2*]

1 Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin, Germany, 2 Machine Learning Group, Technische Universität Berlin, Berlin, Germany, 3 Charité University Hospital, Berlin, Germany, 4 Department of Brain and Cognitive Engineering, Korea University, Seoul, Korea, 5 ISTD Pillar, Singapore University of Technology and Design (SUTD), Singapore

☯ These authors contributed equally to this work.
* sebastian.bach@hhi.fraunhofer.de (SB), klaus-robert.mueller@tu-berlin.de (KM), wojciech.samek@hhi.fraunhofer.de (WS)

CrossMark click for updates

OPEN ACCESS

Citation: Bach S, Binder A, Montavon G, Klauschen F, Müller K-R, Samek W (2015) On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. PLoS ONE 10 (7): e0130140. doi:10.1371/journal.pone.0130140

Abstract

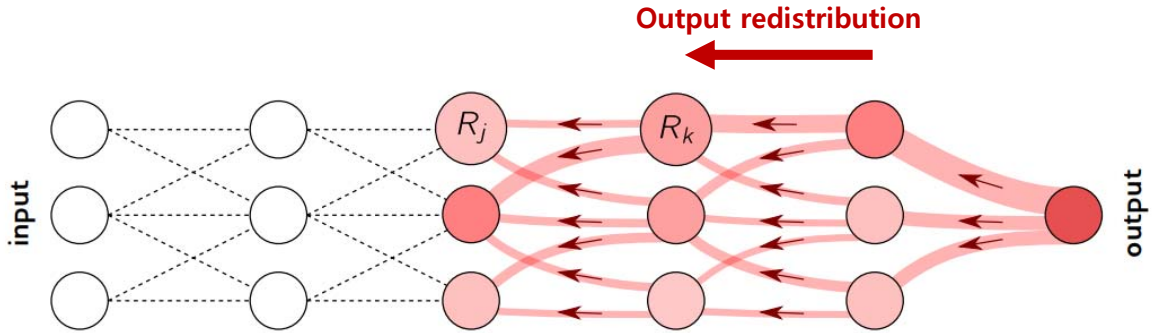
Understanding and interpreting classification decisions of automated image classification systems is of high value in many applications, as it allows to verify the reasoning of the system and provides additional information to the human expert. Although machine learning methods are solving very successfully a plethora of tasks, they have in most cases the dis-

Key Idea

- Decomposition
  - 테일러 시리즈를 활용하여 학습이 완료된 인공지능 모델의 출력 결과를 분해

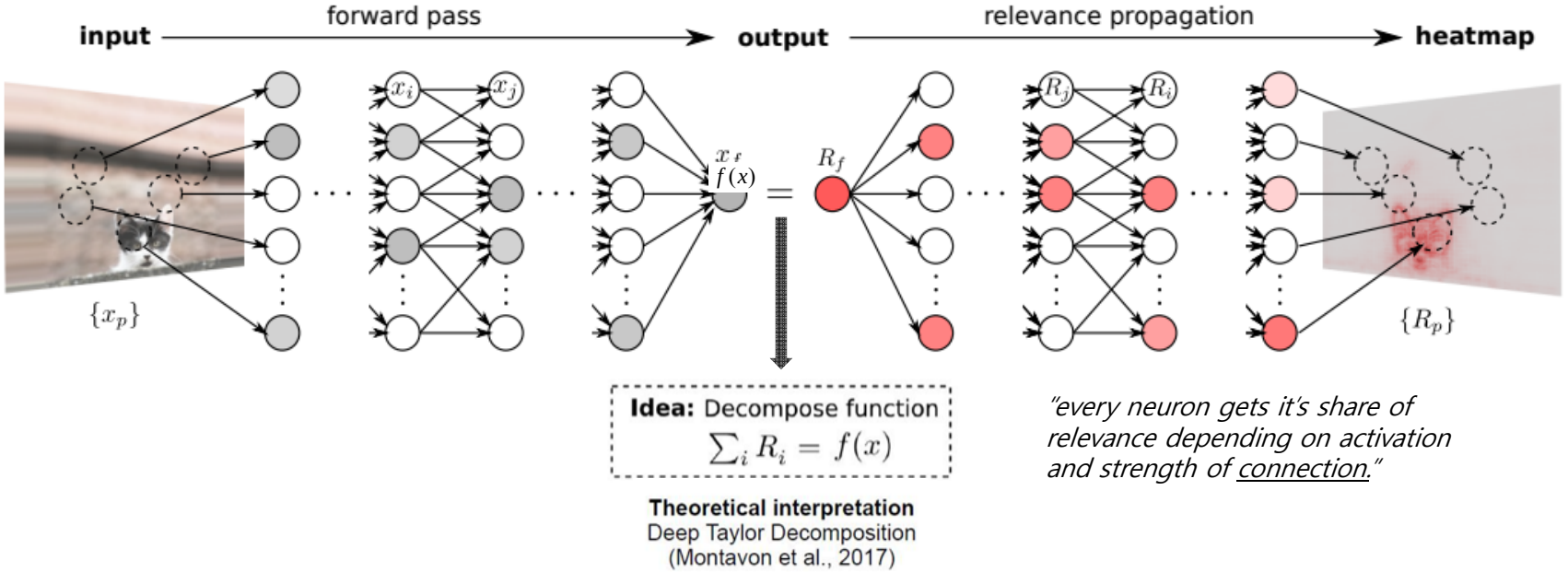
$$f(x) = \sum_{i=1}^{d} R_i$$

- Relevance propagation
  - 각각의 뉴런은 마지막 층의 출력값에 기여하는 **기여도**(**Relevance score**)를 가지고 있음
  - 출력값에서부터 시작하여 기여도를 이전 층 방향으로 **재분배**(**Redistribution**)
  - 정답에 기여하는 기여도를 입력층에 이를 때까지 반복하여 재분배
  - 재분배 시 기여도는 각 층에서 일정한 값으로 보존



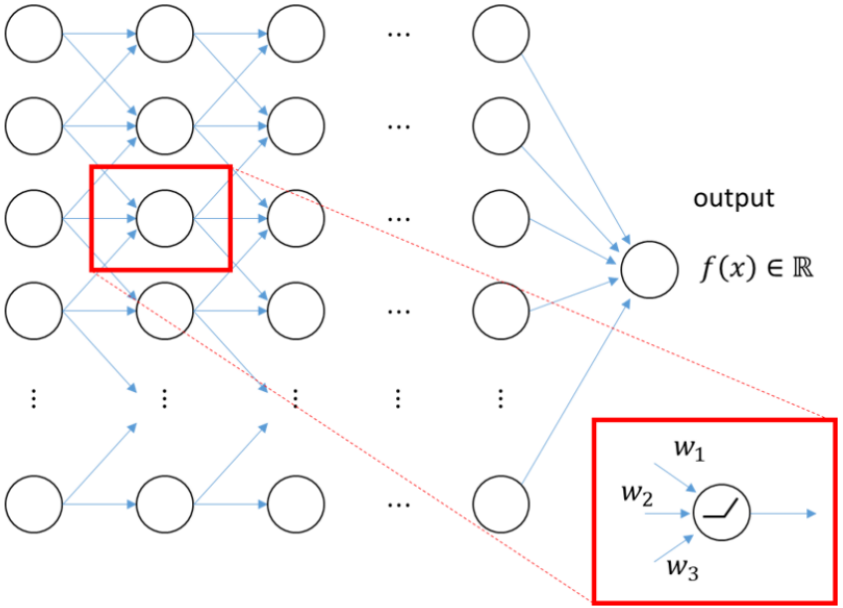
Theoretical Interpretation

- Montavon et al., “Explaining nonlinear classification decisions with deep Taylor decomposition,” Pattern Recognition, vol. 65, pp. 211-222, 2017.



How to Decompose Output?

- 신경망 마지막 단에서 출력값을 어떻게 분해할 것인가?
  - 각 뉴런의 출력값을 수학적으로 분해하여 '기여도'를 정의
  - 테일러 시리즈



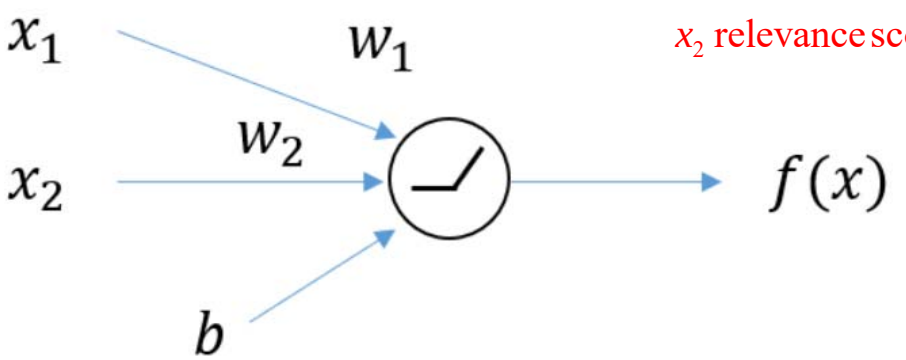
How to Decompose Output?

- 기여도는 각 입력이 출력에 주는 영향을 의미
- 다시 말해, 입력의 변화에 따른 출력의 변화 정도
  - 변화량 = 미분
  - $f(x)$를 $\frac{\partial f}{\partial x_i}$로 분해할 수 있는 이론이 필요함.

→ ***테일러 시리즈***

x_1 relevance score: $\frac{\partial f}{\partial x_1}$

x_2 relevance score: $\frac{\partial f}{\partial x_2}$



2차원 입력과 1차원 출력을 갖는 뉴런

Taylor Series

- 임의의 매끄러운 함수 $f(x)$에 대해 실수 a를 기준으로한 $f(x)$의 테일러 시리즈:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$
$$= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

- 2차 이상의 고차항을 ε으로 치환한 1차 테일러 시리즈:

$$= f(a) + \frac{d}{dx} f(x)\Big|_{x=a} (x-a) + \varepsilon$$

x의 변화에 따른 $f(x)$의 변화량 = 기여도

Multivariate Tayler Series

- 신경망 입력 값은 복수이므로, 다변수 테일러 시리즈를 활용

Example in two dimensions [edit]

For example, the third-order Taylor polynomial of a smooth function $f: \mathbf{R}^2 \to \mathbf{R}$ is, denoting $\boldsymbol{x} - \boldsymbol{a} = \boldsymbol{v}$,

$$P_3(\boldsymbol{x}) = f(\boldsymbol{a}) + \frac{\partial f}{\partial x_1}(\boldsymbol{a})v_1 + \frac{\partial f}{\partial x_2}(\boldsymbol{a})v_2 + \frac{\partial^2 f}{\partial x_1^2}(\boldsymbol{a})\frac{v_1^2}{2!} + \frac{\partial^2 f}{\partial x_1 \partial x_2}(\boldsymbol{a})v_1 v_2 + \frac{\partial^2 f}{\partial x_2^2}(\boldsymbol{a})\frac{v_2^2}{2!} + \frac{\partial^3 f}{\partial x_1^3}(\boldsymbol{a})\frac{v_1^3}{3!} + \frac{\partial^3 f}{\partial x_1^2 \partial x_2}(\boldsymbol{a})\frac{v_1^2 v_2}{2!} + \frac{\partial^3 f}{\partial x_1 \partial x_2^2}(\boldsymbol{a})\frac{v_1 v_2^2}{2!} + \frac{\partial^3 f}{\partial x_2^3}(\boldsymbol{a})\frac{v_2^3}{3!}$$

- 2차 이상 고차항을 ε으로 치환한 1차 다변수 테일러 시리즈

$$f(x) = f(a) + \sum_{p=1}^{d} \frac{\partial f}{\partial x_p}\Big|_{x=a} (x-a) + \varepsilon$$

Deep Taylor Decomposition

- $\varepsilon = 0$ 이라고 가정할 수 있다면
- $f(a) = 0$인 a를 수학적으로 찾을 수 있다면

$$f(x) = f(a) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i = a_i} (x_i - a_i) + \varepsilon$$

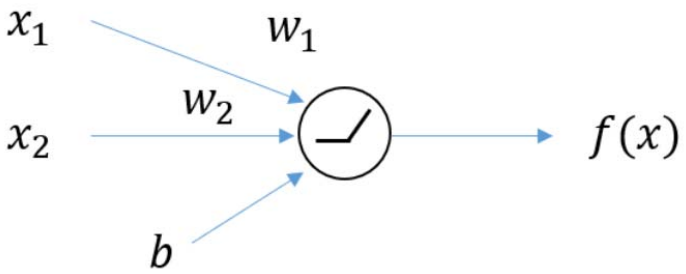
$$= \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i = a_i} (x_i - a_i)$$

$$= \sum_{i=1}^{d} R_i$$

출력값 $f(x)$를 입력값의 기여도로 표현 가능

Deep Taylor Decomposition

- $\varepsilon = 0$ 이라고 가정할 수 있다면
  - 입력 두개와 출력 하나를 갖고, 활성화 함수가 ReLU인 뉴런
  - $\sum_{i=1}^{2} w_i x_i + b > 0$ 인 경우에 대해서만 분석하면 됨



$$f(x) = \max(0, \sum_{i=1}^{2} w_i x_i + b) = \max(0, w_1 x_1 + w_2 x_2 + b)$$

1) if $\sum_{i=1}^{2} w_i x_i + b \leq 0$ $\quad f(x) = 0$

2) if $\sum_{i=1}^{2} w_i x_i + b > 0$ $\quad f(x) = \sum_{i=1}^{2} w_i x_i + b$

Deep Taylor Decomposition

- $\varepsilon = 0$ 이라고 가정할 수 있다면,

$$f(x) = f(a) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i=a_i} (x_i - a_i) + \varepsilon = f(a) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i=a_i} (x_i - a_i)$$

  - 앞서 딥러닝 모델 아키텍쳐에서 $f(x) = w_1 x_1 + w_2 x_2 + b$ 이므로,

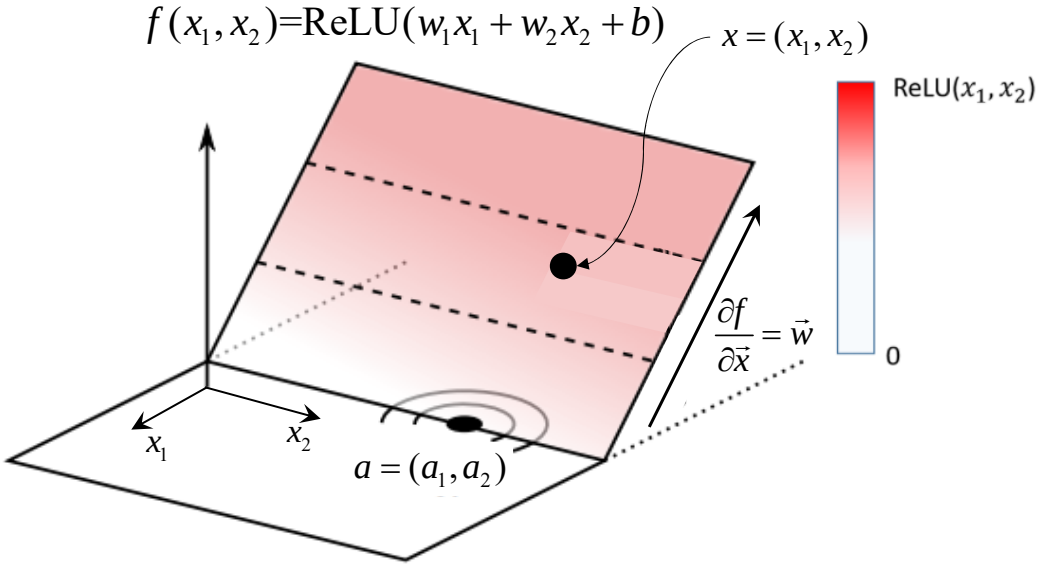
$$\frac{\partial f(x)}{\partial x_1} = w_1, \frac{\partial f(x)}{\partial x_2} = v \quad \frac{\partial^2 f(x)}{\partial x_1^2} = 0, \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} = 0, \cdots$$ 2차 이상의 편미분 계수는 모두 0

  - 따라서 $\varepsilon = 0$ 임을 알 수 있다. 식을 다시 쓰면,

$$f(x) = \sum_{i=1}^{2} w_i x_i + b = f(a) + \sum_{i=1}^{2} w_i (x_i - a_i)$$

Deep Taylor Decomposition

- $f(a) = 0$인 a를 수학적으로 찾을 수 있다면
  - a는 테일러 급수에서 함수를 근사하기 시작하는 점
  - a를 '적절하게' 선정하여 $f(a) = 0$을 만족해야 함



하나의 뉴런에서, 두 개의 입력값과 ReLU 통과 후의 출력값을 도시한 그림

Deep Taylor Decomposition

- $f(a) = 0$인 a를 수학적으로 찾을 수 있다면
  - 입력 데이터 x에 대해 gradient를 따라 t만큼 내려가다 보면 a와 만난다

$$\vec{a} = \vec{x} + t\vec{w} \quad \cdots \quad (1)$$

$$f(\vec{a}) = \sum_i w_i a_i + b = 0 \quad \cdots \quad (2)$$

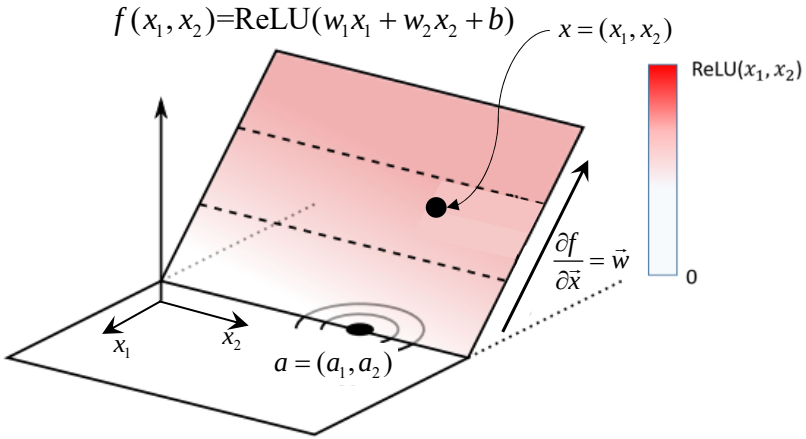
  - 식 (1)을 (2)에 대입하면,

$$f(\vec{a}) = \sum_i w_i (x_i + t w_i) + b = 0$$

$$\sum_i w_i x_i + t \sum_i w_i^2 + b = 0$$

  - 위 식을 t에 대해 전개하면,

$$\therefore t = -\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \quad \cdots \quad (3)$$



Deep Taylor Decomposition

- $f(a) = 0$인 a를 수학적으로 찾을 수 있다면
  - (3)을 (1)에 대입하면,

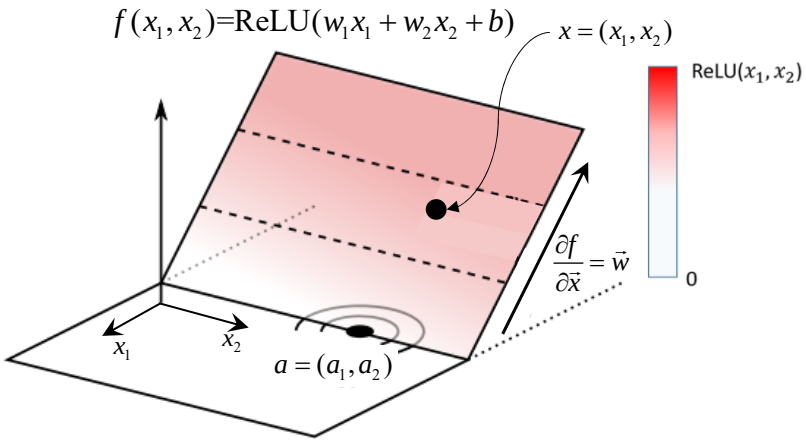
$$a_i = x_i - \left(\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \right) w_i \quad \cdots \quad (4)$$

  - (4)를 통해 $f(x)$를 다음과 같이 표현

$$f(x) = 0 + \sum_i w_i \left(\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \right) w_i + 0$$

$$= \sum_i \left(\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \right) w_i^2$$

$$= \sum_i R_i$$



→ 출력값 $f(x)$를 입력값의 기여도로 표현이 가능!

Deep Taylor Decomposition

- $\varepsilon = 0$ 이라고 가정할 수 있다면 → Ok
- $f(a) = 0$인 a를 수학적으로 찾을 수 있다면 → Ok

$$f(x) = 0 + \sum_i w_i \left(\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \right) w_i + 0 = \sum_i \left(\frac{\sum_i w_i x_i + b}{\sum_i w_i^2} \right) w_i^2 = \sum_i R_i$$

*w, x, b*는 주어진 값이므로, '기여도'를 계산할 수 있다.

Application to Deep Network

- 여러 층을 가지고 있는 모델에 적용하게 되면,
  - 각 층의 입력은 이전 층에서 계산된 기여도
  - 각 층의 출력은 다음 층의 기여도

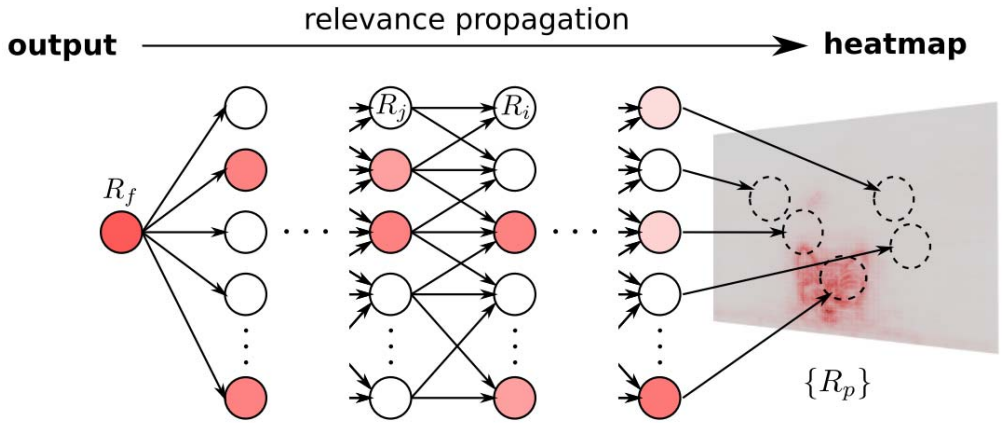
$$f(x) = f(a) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i=a_i} (x_i - a_i) + \varepsilon$$

$$= \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}|_{x_i=a_i} (x_i - a_i)$$

$$= \sum_{i=1}^{d} R_i$$

$$R_i = \sum_j \frac{\partial R_j}{\partial x_i}|_{x_i=a_j} (x_i - a_j)$$

$$R_i = \sum_j R_{ij}$$



Relevance 'propagation'이 수식적으로 가능

Z^+-Rules

- Constrained Input Space and the z^+-Rules
  - 입력 데이터의 도메인이 한정된 공간인 경우: ReLU로 인해 양수 값으로 한정됨

$$R_i = \sum_j \frac{\partial R_j}{\partial x_i}\Big|_{x_i = a_j} (x_i - a_j)$$

$$= w_{ij} \cdot \frac{x_i w_{ij}^+}{w_{ij} \sum_i x_i w_{ij}^+} (\sum_i x_i w_{ij} + b_j)$$

$$= \frac{x_i w_{ij}^+}{\sum_i x_i w_{ij}^+} R_j$$

$$= \frac{z_{ij}^+}{\sum_i z_{ij}^+} R_j$$

where $z_{ij}^+ = x_i w_{ij}^+$ and w_{ij}^+ denotes the positive part of w_{ij}

Demo

TensorFlow Implementation

- Element-wise 계산

```
31      ################ 각 Layer type에 맞는 Relevance score 계산 함수 #########
32      # Fully connected layer의 Relevance를 계산하기 위한 함수
33      def backprop_fc(self, w, b, a, r):
34          z = K.dot(a, K.constant(w)) + b + self.epsilon
35          s = r/z
36          c = K.dot(s, K.transpose(w))
37          return a*c
```

LRP propagation rule:

$$R_j^{(l)} = a_j \sum_k \frac{w_{jk}^+}{\sum_j a_j w_{jk}^+ + b_k^+} R_k^{(l+1)}$$

$$z_k \leftarrow \varepsilon + \sum_j a_j w_{jk}^+$$

$$s_k \leftarrow R_k / z_k$$

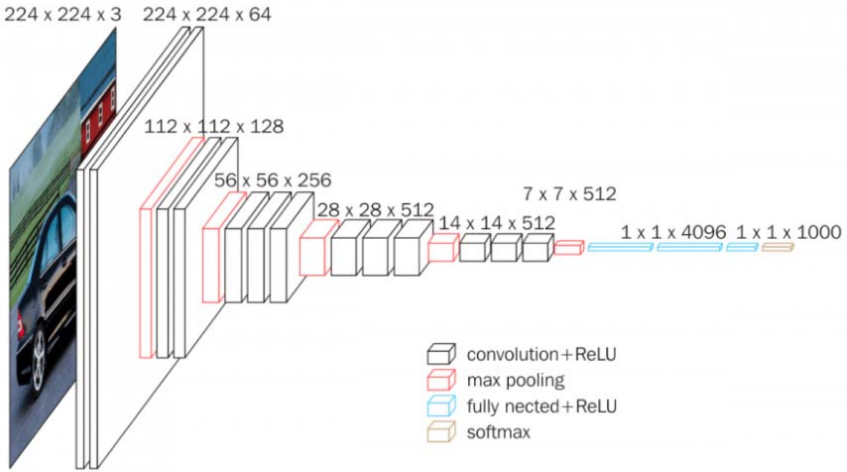
$$c_j \leftarrow \sum_k w_{jk}^+ s_k$$

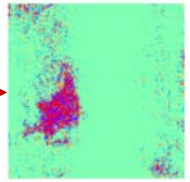
$$R_j \leftarrow a_j c_j$$

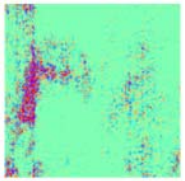
Montavon, G., et al., *Explaining nonlinear classification decisions with deep taylor decomposition.* 2017. **65**: p. 211-222.

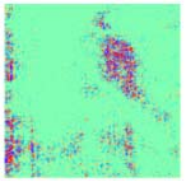
Implementation

- Pre-trained VGG16




Label predict → "Dog" → LRP



Label predict → "Cat" → LRP



Label predict → "Rabit?" → LRP


Import Library

- 라이브러리 호출

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import os
from matplotlib.cm                import get_cmap


from tensorflow.keras.preprocessing.image    import img_to_array, load_img
from tensorflow.keras.applications.vgg16     import (preprocess_input, decode_predictions)
from tensorflow.keras                        import backend as K
from tensorflow.python.ops                   import gen_nn_ops
from tensorflow.keras.applications.vgg16     import VGG16

tf.compat.v1.disable_eager_execution()
```

Load Data

- ImageNet 데이터 로드

```
processed_images = []
image_ = load_img('./images/cat.25.jpg', target_size=(224, 224))

plt.imshow(image_)
plt.axis('off')

# 학습에 사용할 이미지를 3차원 배열로 변환 및 전처리하는 코드
# processed_images에는 학습에 사용할 이미지가 들어가게 된다.

image = img_to_array(image_)
image = preprocess_input(image)
processed_images.append(image)
processed_images = np.array(processed_images)
```

출력

Functions

- 함수 선언

```python
# Pretrain 된 VGG16의 layer name, activation, weight에 대한 정보를 불러온다.
def get_model_params(model):
    names, activations, weights = [], [], []
    for layer in model.layers:
        name = layer.name if layer.name != 'predictions' else 'fc_out'
        names.append(name)
        activations.append(layer.output)
        weights.append(layer.get_weights())
    return names, activations, weights

# Heatmap을 그리기 위한 함수
def heatmap(image, cmap_type='rainbow', reduce_op='sum', reduce_axis=-1, **kwargs):
    cmap = get_cmap(cmap_type)
    shape = list(image.shape)

    # RGB 세 채널의 결과가 계산되므로, 세 채널의 LRP 결과를 모두 합하여 출력한다.
    reduced_image = image.sum(axis=-1)

    # LRP 출력 결과를 0~1 사이 범위의 값으로 normalize한다.
    projected_image = project_image(reduced_image, **kwargs)
    heatmap = cmap(projected_image.flatten())[:, :3].T
    heatmap = heatmap.T
    shape[reduce_axis] = 3
    return heatmap.reshape(shape)
```

Pre-trained VGG16의 layer name, activation, weight에 대한 정보를 불러오는 함수

출력된 LRP 결과값을 heatmap으로 표현하기 위한 함수

Functions

- 함수 선언

```
26  # LRP 출력 결과를 normaliize하는 함수
27  def project_image(image, output_range=(0, 1), absmax=None, input_is_positive_only=False):
28      if absmax is None:
29          absmax = np.max(np.abs(image), axis=tuple(range(1, len(image.shape))))
30      absmax = np.asarray(absmax)
31      mask = (absmax != 0)
32      if mask.sum() > 0:
33          image[mask] /= absmax[mask]
34      if not input_is_positive_only:
35          image = (image + 1) / 2
36      image = image.clip(0, 1)
37      projection = output_range[0] + image * (output_range[1] - output_range[0])
38      return projection
```

출력된 LRP 결과값을 0과 1 사이로 normalize 하는 함수

LRP Code

- LRP 클래스 선언

```
1  class LayerwiseRelevancePropagation:
2
3      def __init__(self, epsilon=1e-7):
4
5          self.model = VGG16(include_top=True, weights='imagenet', input_shape=(224, 224, 3))
6          self.epsilon = epsilon
7          self.names, self.activations, self.weights = get_model_params(self.model)
8          self.num_layers = len(self.names)
9          self.relevance = self.compute_relevances()
10         self.lrp_runner = K.function(inputs=[self.model.input, ], outputs=[self.relevance, ])
```

LRP 클래스 선언

- model: pretrained VGG16 값이 할당됨
- epsilon: 발산을 막기 위한 값
- names, activations, weights: VGG16의 layer 이름, activation 값, weights
- num_layers: VGG16의 layer 개수
- relevance: LRP의 relevance를 계산하기 위한 객체
- Lrp_runner: LRP를 실행하는 함수

Compute Relevance

- LRP 클래스 선언

```
12      # 각 Layer의 relevance를 계산하는 함수
13      def compute_relevances(self):
14          r = self.model.output
15          for i in range(self.num_layers-2, -1, -1):
16              if 'fc' in self.names[i + 1]:
17                  r = self.backprop_fc(self.weights[i + 1][0], self.weights[i + 1][1], self.activations[i], r)
18              elif 'flatten' in self.names[i + 1]:
19                  r = self.backprop_flatten(self.activations[i], r)
20              elif 'pool' in self.names[i + 1]:
21                  r = self.backprop_max_pool2d(self.activations[i], r)
22              elif 'conv' in self.names[i + 1]:
23                  r = self.backprop_conv2d(self.weights[i + 1][0], self.weights[i + 1][1], self.activations[i], r)
24              else:
25                  raise 'Layer not recognized!'
26                  sys.exit()
27          return r
```

VGG16은 Dense layer 뿐만 아니라, flatten, maxpooling, convolution layer가 존재한다. 각 layer 종류에 맞게 Relevance score를 계산하기 위한 함수

Relevance Score

- 각 층 종류에 맞는 Relevance score 계산 방법
  - Fully connected layer, flatten layer

```
29    # Fully connected layer의 Relevance를 계산하기 위한 함수
30    def backprop_fc(self, w, b, a, r):
31
32        z = K.dot(a, K.constant(w)) + b + self.epsilon
33        s = r/z
34        c = K.dot(s, K.transpose(w))
35        return a*c
36
37    # Flatten layer의 Relevance를 넘겨주기 위해 입력 모양을 바꾸는 함수
38    def backprop_flatten(self, a, r):
39        shape = a.get_shape().as_list()
40        shape[0] = -1
41        return K.reshape(r, shape)
```

값의 발산을 막기 위해 epsilon 값을 넣어준다.

$$z_k \leftarrow \varepsilon + \sum_j a_j w_{jk}^+$$

$$s_k \leftarrow R_k / z_k$$

$$c_j \leftarrow \sum_k w_{jk}^+ s_k$$

$$R_j \leftarrow a_j c_j$$

Relevance Score

- 각 층 종류에 맞는 Relevance score 계산 방법
  - Maxpooling layer, convolution layer

```
43      # Maxpooling layer의 Relevance를 계산하기 위한 함수
44      def backprop_max_pool2d(self, a, r, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1)):
45          z = K.pool2d(a, pool_size=ksize[1:-1], strides=strides[1:-1], padding='valid', pool_mode='max')+self.epsilon
46          s = r/z
47          c = gen_nn_ops.max_pool_grad_v2(a, z, s, ksize, strides, padding='VALID')
48          return a*c
49
50      # Convolution layer의 Relevance를 계산하기 위한 함수
51      def backprop_conv2d(self, w, b, a, r, strides=(1, 1, 1, 1)):
52          z = K.conv2d(a, kernel=w, strides=strides[1:-1], padding='same') + b + self.epsilon
53          s = r/z
54          c = tf.compat.v1.nn.conv2d_backprop_input(K.shape(a), w, s, strides, padding='SAME')
55          return a*c
```

$$z_k \leftarrow \varepsilon + \sum_j a_j w_{jk}^+$$

$$s_k \leftarrow R_k / z_k$$

$$c_j \leftarrow \sum_k w_{jk}^+ s_k$$

$$R_j \leftarrow a_j c_j$$

Additional Functions

- 함수 선언

```
57      # 입력 데이터의 정답을 불러오는 함수
58      def predict_labels(self, images):
59          preds = self.model.predict(images)
60          decoded_preds = decode_predictions(preds)
61          labels = [p[0] for p in decoded_preds]
62          return labels
63
64      # 아래 함수를 통해 lrp를 계산한다
65      def run_lrp(self, images):
66          print("Running LRP on {0} images...",format(len(images)))
67          return self.lrp_runner([images, ])[0]
68
69      # lrp 결과를 heatmap으로 표현하는 함수
70      def compute_heatmaps(self, images, g=0.2, cmap_type='rainbow', **kwargs):
71          lrps = self.run_lrp(images)
72          print("LRP run successfully...")
73
74          heatmaps = [heatmap(lrp, cmap_type=cmap_type, **kwargs) for lrp in lrps]
75
76          return heatmaps
```

VGG16에서 출력된 라벨을 저장하기 위한 함수 labels 변수에는 클래스 이름, 클래스 설명, 스코어에 대한 정보가 들어간다.

앞에서 선언한 LRP 함수를 실행하는 함수

Heatmap을 계산하기 위한 함수

Run LRP

- LRP 실행

```
# lrp 객체 생성
lrp = LayerwiseRelevancePropagation()

# 입력 데이터의 정답을 불러온다.
labels = lrp.predict_labels(processed_images)
print("Labels predicted...")

# 입력 데이터의 lrp를 계산한 후 heatmap을 그린다.
heatmaps = lrp.compute_heatmaps(processed_images)
print("Heatmaps generated...")
```

앞에서 선언한 LRP 객체를 생성하고, label과 heatmap을 계산한다.

출력

```
Labels predicted...
Running LRP on 1 images...
LRP run successfully...
Heatmaps generated...
```

Results

```python
# irp 계산 결과 출력
fig = plt.figure()
fig.suptitle("Class name: %s \n Class description: %s \n Score: %.4f" %(labels[0][0], labels[0][1], labels[0][2]),#
             fontsize=15)

ax0 = fig.add_subplot(121)
ax0.axis('off')
ax0.imshow(image_)

ax1 = fig.add_subplot(122)
ax1.axis('off')
ax1.imshow(heatmaps[0], interpolation='bilinear')

plt.show()
```

출력

Class name: n02124075
Class description: Egyptian_cat
Score: 0.5813