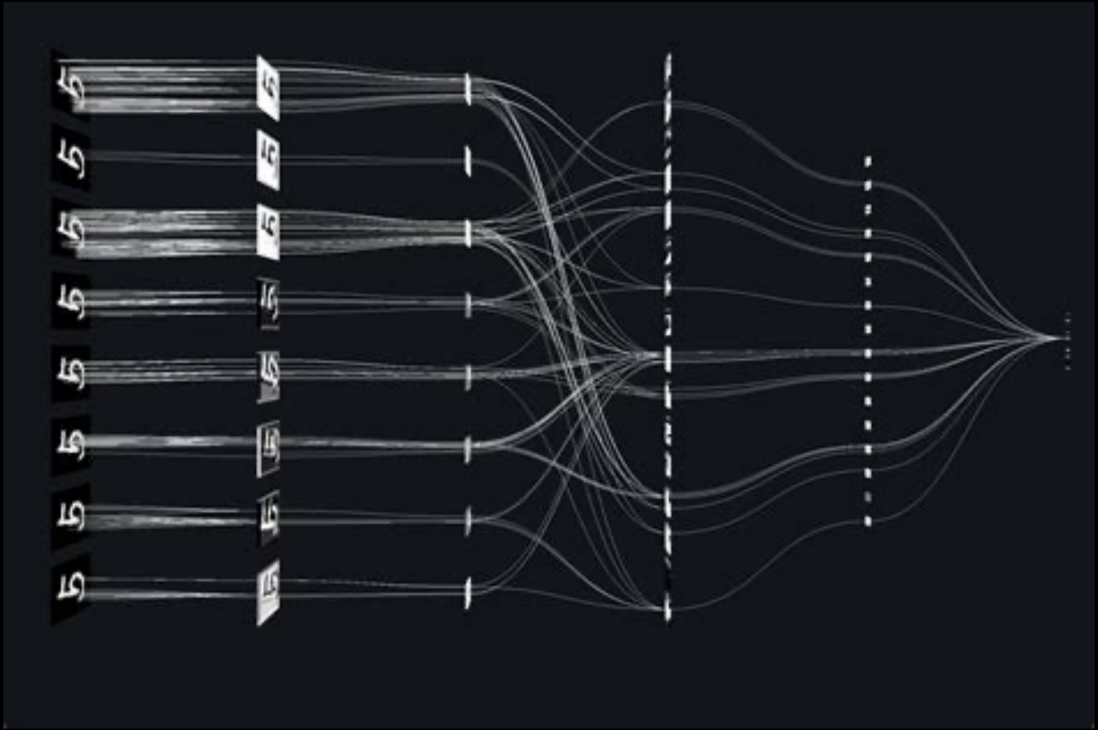
Explainable ML - XML

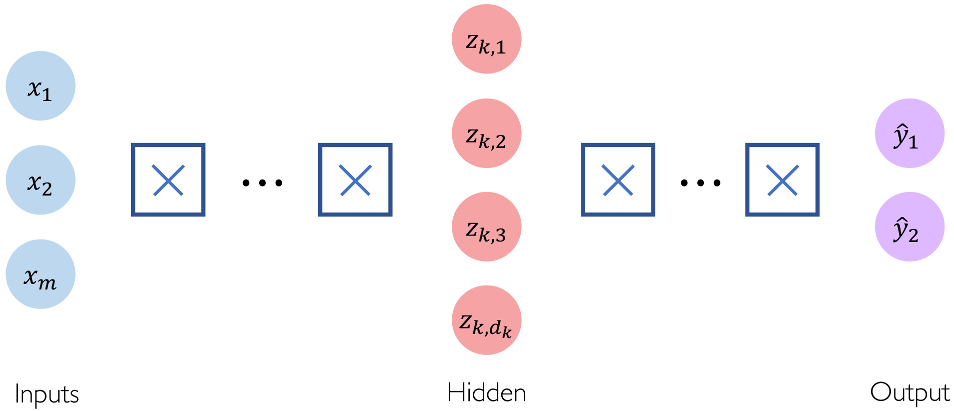
Pierre Gentine – Columbia University



The problem

Major advances with the deep learning

Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Many (many) weights → black box

How to open up the black box?

Interpreting what is in the gut of deep NN/CNN

Explainable ML

Also helps build trustworthiness in models

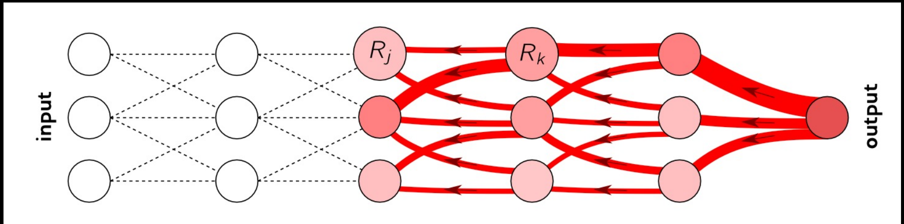
→ Right answer for the right reason

A few examples:

1. Layerwise Relevance Propagation (LRP)
2. Adjoint/gradient – Saliency maps

1. Layerwise Relevance Propagation (LRP)

Trying to assess where the information is coming from
Backward – from output to input that most explain the output

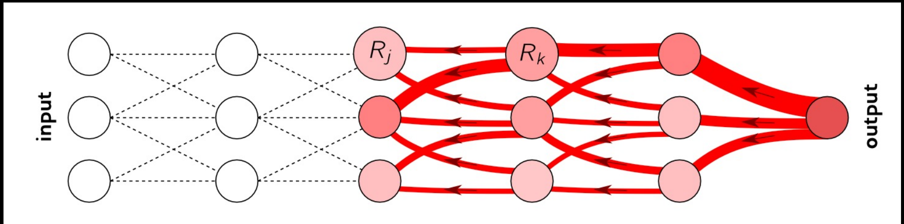


Pass information/relevance backward – a la Kirchoff law (conservation of current):

Toms et al. 2020 JAMES

1. Layerwise Relevance Propagation (LRP)

Trying to assess where the information is coming from
Backward – from output to input that most explain the output



Pass information/relevance backward – a la Kirchoff law (conservation of current):

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

Layer *j* before layer *k*
a_j – output from layer *j* (after activation function)
w_{jk} - weights

Toms et al. 2020 JAMES

1. Layerwise Relevance Propagation (LRP)

Different LRP rules

1. LRP-0

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

Equivalent to gradient x input

2. LRP-epsilon

Limit noisy information – filter by small noise epsilon

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

3. LRP-gamma

Favor positive contributions compared to negative contributions

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

Montavon et al.

1. Layerwise Relevance Propagation (LRP)

Different LRP rules → generic rule

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

Can split computation into 4 substeps

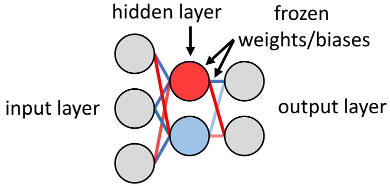
$$\begin{aligned}
&\forall_k: \ z_k = \epsilon + \textstyle\sum_{0,j} a_j \cdot \rho(w_{jk}) && \text{(forward pass)}\\
&\forall_k: \ s_k = R_k / z_k && \text{(element-wise division)}\\
&\forall_j: \ c_j = \textstyle\sum_k \rho(w_{jk}) \cdot s_k && \text{(backward pass)}\\
&\forall_j: \ R_j = a_j \, c_j && \text{(element-wise product)}
\end{aligned}$$

What we will do in the notebook – stay tuned

Montavon et al.

1. Layerwise Relevance Propagation (LRP)

1) Train network & freeze weights/biases

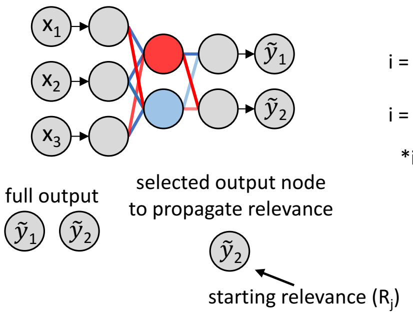


Information learned during training:

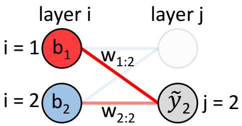
positive weights/biases

negative weights/biases

2) Input sample into frozen network and retain output



3) Propagate relevance from output node to previous layer



*ignored nodes/weights are transparent

propagation rule

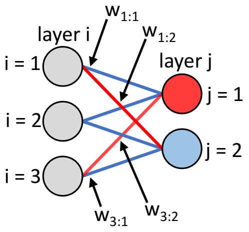
$$R_i = \sum_j \frac{a_i w_{ij}^{+} + \max(0, b_j)}{\sum_i a_i w_{ij}^{+} + max(0, b_j)} R_j$$

example relevance calculations

$$R_{i=1} = \left(\frac{a_1 w_{1:2}}{a_1 w_{1:2} + a_2 w_{2:2}}\right) \tilde{y}_2$$

$$R_{i=2} = \left(\frac{a_2 w_{2:1}}{a_1 w_{1:2} + a_2 w_{2:2}}\right) \tilde{y}_2$$

4) Propagate relevance from hidden layer to input layer



propagation rule

$$R_i = \sum_j \left(\frac{w_{ij}^2}{\sum_i w_{ij}^2}\right) R_j$$

*all biases are ignored for this rule

relevance at input layer

R_1 R_2 R_3

example relevance calculation

$$R_{i=1} = \left(\frac{{w_{1:1}}^2}{{w_{1:1}}^2 + {w_{2:1}}^2 + {w_{3:1}}^2}\right) R_{j=1} + \left(\frac{{w_{1:2}}^2}{{w_{1:2}}^2 + {w_{2:2}}^2 + {w_{3:2}}^2}\right) R_{j=2}$$

*relevance calculations are similar for other input nodes

5) Repeat for each sample of interest…

Toms et al. 2020 JAMES

1. Layerwise Relevance Propagation (LRP)

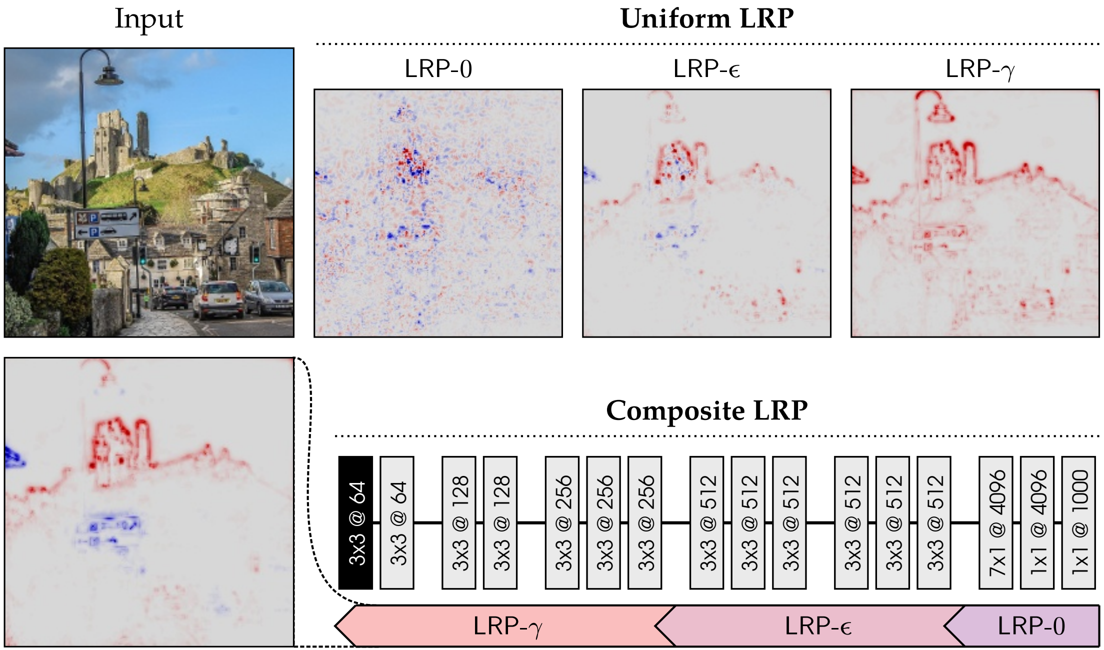


Fig. 10.4. Input image and pixel-wise explanations of the output neuron 'castle' obtained with various LRP procedures. Parameters are $\epsilon = 0.25\,\text{std}$ and $\gamma = 0.25$.

2. Gradient/adjoint

Gradient/adjoint of output **y** to inputs **x**

$$\nabla_{\mathbf{x}}\mathbf{y}$$

$$J = \begin{pmatrix} \frac{\partial \mathbf{y}}{\partial x_1} & \cdots & \frac{\partial \mathbf{y}}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

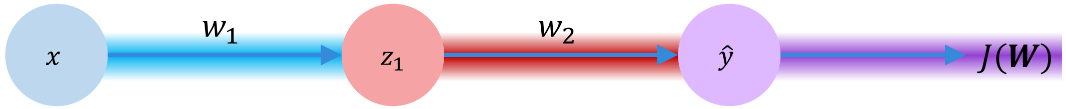
2. Gradient/adjoint

Gradient/adjoint of output **y** to inputs **x**

$$J = \begin{pmatrix} \frac{\partial \mathbf{y}}{\partial x_1} & \cdots & \frac{\partial \mathbf{y}}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

Once upon a time: **backpropagation**

Chain's rule



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Same thing!

2. Gradient/adjoint

Personal experience:

Non-linear so average over different samples

Use non-dimensional/normalized inputs (and ideally outputs)

Tends to work well for first cut – but can be noisy