DEBUGGING AND UNDERSTANDING DEEP LEARNING MODELS

Narine Kokhlikyan
Research Scientist
@NARINEK
AGENDA

01
INTRO TO MODEL INTERPRETABILITY AND EXPLAINABILITY

02
EXPLAINING BBX VS BUILDING INHERENTLY INTERPRETABLE MODELS

03
GRADIENT AND PERTURBATION-BASED ATTRIBUTION ALGORITHMS

04
CASE STUDY: SEMANTIC SEGMENTATION

05
CONCEPT-BASED MODEL INTERPRETABILITY

06
MODEL COMPARISON AND CORRELATION ANALYSIS

07
RECAP & THE FUTURE
INTERPRETABILITY VS EXPLAINABILITY

The line between these two concepts is blurry and often ill-defined.

EXPLAINABILITY

“Seeks answers to ‘why question’ about the decisions and behavior of our model.”

INTERPRETABILITY

“Describes AI model internals and their predictions in human understandable terms.”

* LH Gilpin, et. al., Explaining explanations: An overview of interpretability of machine learning in IEEE 5th International Conference on data science and advanced analytics (DSAA), 2018
BLACK BOX MODELS

The Mythos of Model Interpretability, Zachary C. Lipton, 2017
The Mythos of Model Interpretability, Zachary C. Lipton, 2017
The Mythos of Model Interpretability, Zachary C. Lipton, 2017
DESIDERATA OF INTERPRETABILITY*

- Trust
- Causality
- Transferability
- Informativeness
- Fair and ethical decision making

* The Mythos of Model Interpretability, Zachary C. Lipton, 2017
• Is this the confidence about model performance?
• Given an input what prediction can we expect from our model
• Model’s performance can be different in deployment environment
  • We can’t expect that our models always account for all kinds of biases in the data
CAUSALITY

• Inferring causal relationships from observational data
  • e.g. smoking and cancer, thalidomide use and birth defects
TRANSFERABILITY

- Generalizability and transferring knowledge to different domains
  - Ability to adopt to new environments
- ML models are susceptible to adversarial attacks and can be easily fooled
INFORMATIVENESS

- ML models convey information about the prediction only through output value
  - This is not very informative; there should be more descriptive ways of doing it
  - e.g. **Input:** `Where I can buy best chocolate?`  **Answer:** ‘Chocolatier Desiree’
FAIR AND ETHICAL DECISION MAKING

• How can we make sure that our model made decision fairly?
  • e.g. predictions related to recidivism
• Regulations for algorithmic decisions
  • Contesting the propositions and modifying the decisions
EXPLAINING / INTERPRETING BLACK BOX
VS
BUILDING INHERENTLY INTERPRETABLE MODELS
EXPLAINING BLACK BOX MODELS

• Aka Post-Hoc Interpretability

• Infer behavior of a pre-trained model
  • based on perturbed inputs
  • gradient back-propagation
  • visualization

• No sacrifice of predictive performance
BUILDING INHERENTLY INTERPRETABLE MODELS

- By looking at the output of the model we can tell how the model came to a decision
  - e.g. decision trees, simple linear models
- More challenging for Deep Neural Networks (DNNs)
  - Model’s decision making is attributed to a number of prototype samples based on similarity ("This Looks Like That", Chen C, et al., 2019)
- Might require performance sacrifice
EXPLAINING / INTERPRETING BLACK BOX DEEP LEARNING MODELS

GRADIENT AND PERTURBATION - BASED ATTRIBUTION METHODS
GRADIENT-BASED ATTRIBUTION ALGORITHMS

- Describes the infinitesimal change in inputs that changes the output
- Requires model’s forward and backward passes
PERTURBATION-BASED ATTRIBUTION ALGORITHMS

• Observe changes of model output when the inputs are perturbed
  • e. g. Feature Ablation, Permutation, Shapley Values
GRADIENT - BASED ATTRIBUTION METHODS
SALIENCY

- Infinitesimal change in inputs that changes the output

\[
\Phi_c(x) = \frac{\partial F_c(x)}{\partial x}, \text{ where } x \in \mathbb{R}^N \text{ is the input}
\]

\[
F : \mathbb{R}^N \rightarrow \mathbb{R}_c \text{ is the NN function}
\]

\[
c \text{ is the number of classes}
\]

* Deep Inside Convolutional Networks, Simonyan, 2014
SALIENCY

• Infinitesimal change in inputs that changes the output

\[ \Phi_c(x) = \frac{\partial F_c(x)}{\partial x^*}, \text{ where } x \in \mathbb{R}^N \text{ is the input} \]

\[ F : \mathbb{R}^N \rightarrow \mathbb{R}_c \text{ is the NN function} \]

\[ c \text{ is the number of classes} \]

* Deep Inside Convolutional Networks, Simonyan, 2014

Saliency Maps trained on CIFAR10 dataset
SALIENCY

- Infinitesimal change in inputs that changes the output

\[ \Phi_c(x) = \frac{\partial F_c(x)}{\partial x} \], where \( x \in \mathbb{R}^N \) is the input

\[ F : \mathbb{R}^N \rightarrow \mathbb{R}_c \] is the NN function

\( c \) is the number of classes

Limitations

- Gradients are noisy and sensitive to functions input

* Deep Inside Convolutional Networks, Simonyan, 2014
SMOOTHGRAD

• Adds noise to remove noise, SmoothGrad, Smilkov 2017

• Samples n samples in the neighborhood of input x and averages the explanations across all those samples

\[ \hat{\Phi}_c(x) = \frac{1}{n} \sum_{0}^{n} \Phi_c(x + \mathcal{N}(0, \sigma)), \quad \sigma \text{ is std} \]

• Helps to stabilize explanations

Saliency Maps with SmoothGrad trained on CIFAR10 dataset
SMOOTHGRAD

- Adds noise to remove noise, SmoothGrad, Smilkov 2017
- Samples n samples in the neighborhood of input x and averages the explanations across all those samples

\[
\hat{\Phi}_c(x) = \frac{1}{n} \sum_{0}^{n} \Phi_c(x + \mathcal{N}(0,\sigma)), \ \sigma \text{ is std}
\]

- Helps to stabilize explanations

Limitations

- Finding optimal n can be challenging
- Computationally expensive depending on how large n is

Saliency Maps with SmoothGrad trained on CIFAR10 dataset
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et al., 2017)
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)

$$\Phi(x^i) = (x^i - x_0^i) \cdot \int_0^1 \frac{\partial F(x_0^i + \alpha \cdot (x^i - x_0^i))}{dx^i} d\alpha,$$

where $\alpha \in [0,1]$ is a scaling factor
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)

$$
\Phi(x^i) = (x^i - x_0^i) \cdot \int_0^1 \frac{\partial F(x_0^i + \alpha \cdot (x^i - x_0^i))}{dx^i} d\alpha,
$$

where $\alpha \in [0,1]$ is a scaling factor.

**Limitations**

- Finding good baselines can be challenging
- Feature correlations and interactions aren’t taken into account
AXIOMS OF INTEGRATED GRADIENTS

• Completeness
  • Sum of the attributions is equal to the Neural Network (NN) function’s differences at its input and baseline

\[ \sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x') \]
AXIOMS OF INTEGRATED GRADIENTS

• Completeness
  
  • Sum of the attributions is equal to the Neural Network (NN) function’s differences at its input and baseline

\[
\sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x')
\]

• Sensitivity
  
  • Sensitive to differing features and output predictions
AXIOMS OF INTEGRATED GRADIENTS

- Completeness
  - Sum of the attributions is equal to the Neural Network (NN) function’s differences at its input and baseline
  \[ \sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x') \]

- Sensitivity
  - Sensitive to differing features and output predictions

- Implementation Invariance
  - Attributions for two functionally equivalent NNs are identical
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution

Limitations

- Most of these methods except DeepLIFT are insensitive to parameter randomization (When Explanations Lie, Sixt 2020)
PERTURBATION - BASED ATTRIBUTION METHODS
FEATURE ABLATION

• Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated
FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

<table>
<thead>
<tr>
<th>input ( x \in \mathbb{R}^N )</th>
<th>mask ( m \in \mathbb{Z}^{0+} )</th>
<th>baseline ( b \in \mathbb{R}^N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.29 0.51 0.25 0.68</td>
<td>0 1 0 2</td>
<td>0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>0.07 0.86 0.13 0.10</td>
<td>0 1 0 2</td>
<td>1.0 -1.0 1.0 0.0</td>
</tr>
<tr>
<td>0.18 0.72 0.31 0.68</td>
<td>0 1 0 2</td>
<td></td>
</tr>
<tr>
<td>0.07 0.31 0.31 0.40</td>
<td>0 1 0 2</td>
<td></td>
</tr>
</tbody>
</table>
FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

\[ \Phi(F, x, m, b) = F(x) - F(x, m, b), \]
where \( m \in \mathbb{Z}^{0+} \) is the ablation mask and \( b \in \mathbb{R}^N \) ablation values
FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated.

- \( \Phi(F, x, m, b) = F(x) - F(x, m, b) \), where \( m \in \mathbb{Z}^{0+} \) is the ablation mask and \( b \in \mathbb{R}^{N} \) ablation values.

- We can also combine it with a loss function or any evaluation metric.

\[ \Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t), \quad \text{where} \quad l : \mathbb{R}^{N} \rightarrow \mathbb{Z}^{M} \] is the loss and \( t \in \mathbb{Z}^{M} \) the labels.
FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

\[ \Phi(F, x, m, b) = F(x) - F(x, m, b), \text{ where } m \in \mathbb{Z}^{0+} \text{ is the ablation mask and } b \in \mathbb{R}^N \text{ ablation values} \]

- We can also combine it with a loss function or any evaluation metric

\[ \Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t), \text{ where } l : \mathbb{R}^N \rightarrow \mathbb{Z}^M \text{ is the loss and } t \in \mathbb{Z}^M \text{ the labels} \]

- Occlusion (Visualizing and Understanding Convolutional Networks, Zeiler, et. al., 2013)

  - Ablates rectangular patches of inputs and computes the differences of output function with and without ablation
FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

\[ \Phi(F, x, m, b) = F(x) - F(x, m, b) \]

\[ m \in \mathbb{Z}^{0+} \text{ is the ablation mask and } b \in \mathbb{R}^N \text{ ablation values} \]

- We can also combine it with a loss function or any evaluation metric

\[ \Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t) \]

\[ l : \mathbb{R}^N \rightarrow \mathbb{Z}^M \text{ is the loss and } t \in \mathbb{Z}^M \text{ the labels} \]

Limitations

- Identifying which features mask together and what ablation values use

- Inputs with ablated features might be out of test/train/valid data distributions
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>...</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0.23</td>
<td>5.3</td>
<td>...</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Input: $x \downarrow$

Interpretable binary repr. of $x$: $x' \rightarrow$

[0 1 ... 1]
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree (Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016)

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>...</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0.23</td>
<td>5.3</td>
<td>...</td>
<td>4.0</td>
</tr>
<tr>
<td>x'</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Input $x$ → Interpretable binary repr. of $x$ → $x'$ → Perturbed Input in the neighborhood of $x'$
LIME

• Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]

\[
\begin{array}{cccc}
\text{F1} & \text{F2} & \ldots & \text{FN} \\
0.23 & 5.3 & \ldots & 4.0 \\
0 & 1 & \ldots & 1
\end{array}
\]

Interpretable binary repr. of \(x\) \(\rightarrow\) \(x'\)

Converts interpretable repr. into original repr. \(\rightarrow z = h_z(z')\)

Perturbed Input in the neighborhood of \(x'\)
LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]
LIME

\[
x \rightarrow x' \rightarrow z = h_z(z')
\]

Input \( x \)

Interpretable binary repr. of \( x \)

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>...</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.23</td>
<td>5.3</td>
<td>...</td>
<td>4.0</td>
</tr>
</tbody>
</table>

\[
x' \rightarrow 0 \quad 1 \quad ... \quad 1
\]

Perturbed Input in the neighborhood of \( x' \)

Converts interpretable repr. into original repr. \( z \in Z \)

Compute similarity between \( x \) and \( z \in Z \)

Interpretable model

\[
\pi_x(z)
\]

\[
g(z')
\]
LIME

Input $x$ → $x'$

Interpretable binary repr. of $x$ → $x'$

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between $x$ and $z \in Z$ → $\pi_x(z)$

Train interpretable model $g(z')$. 

Perturbed Input in the neighborhood of $x'$. 

$\begin{array}{cccc}
F1 & F2 & \ldots & FN \\
0.23 & 5.3 & \ldots & 4.0 \\
0 & 1 & \ldots & 1 \\
\end{array}$
LIME

Input $x$ → $x'$

Interpretable binary repr. of $x$ → $x'$

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>...</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.23</td>
<td>5.3</td>
<td>...</td>
<td>4.0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Perturbed Input in the neighborhood of $x'$

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between $x$ and $z \in Z$ → $\pi_x(z)$

Train interpretable model $g(z')$

BBX

$f(z)$
LIME

Interpretable binary repr. of $x$ → $x'$

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between $x$ and $z \in Z$ → $\pi_x(z)$

Train interpretable model $g(z')$

Minimize the weighted distance

$$L(f, g, \pi) = \sum_{z \in Z, z' \in Z'} \pi_x(z)(f(z) - g(z'))^2$$
LIME

Minimizing $L(f, x, g)$ helps us to estimate $w_i^g$ which serve as importance scores for interpretable features

$$L(f, g, \pi) = \sum_{z \in Z, z' \in Z'} \pi_x(z)(f(z) - g(z'))^2$$

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>...</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.23</td>
<td>5.3</td>
<td>...</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Input $x \rightarrow x'$

Interpretable binary repr. of $x$  

Importance scores $\phi \rightarrow w_1^g, w_2^g, ... w_N^g$
LIME

Limitations

- Depends on choice sampling technique, sample size and similarity function
- Depends on the accuracy and complexity of the interpretable model
SHAPLEY VALUES

- Measures expected marginal contributions of each feature for given sample and prediction - a technique adopted from cooperative game theory*

SHAPLEY VALUES

- Measures expected marginal contributions of each feature for given sample and prediction - a technique adopted from cooperative game theory*

### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
<td>$500,000</td>
</tr>
</tbody>
</table>

\[ \text{F(40 years, 3 rooms, SF Downtown)} = 500,000 \]
What are the contributions of each feature for this prediction?

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction?
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Rooms</td>
<td></td>
<td>3 rooms</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td></td>
<td></td>
<td>SF Downtown</td>
</tr>
<tr>
<td>Price</td>
<td></td>
<td></td>
<td>$500.000</td>
</tr>
</tbody>
</table>

$F(40 \text{ years}, 3 \text{ rooms}, \text{SF Downtown}) = 500,000$  

What are the contributions of each feature for this prediction?
What are the contributions of each feature for this prediction?

F(40 years, 3 rooms, SF Downtown) = $500,000
## SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>Feature IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Age</td>
<td>#Rooms</td>
</tr>
<tr>
<td>40 years</td>
<td>3 rooms</td>
</tr>
</tbody>
</table>

\[ F(40 \text{ years}, 3 \text{ rooms}, \text{SF Downtown}) = $500,000 \]

What are the contributions of each feature for this prediction?


**SHAPLEY VALUES**

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Rooms</td>
<td>3 rooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>SF Downtown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>$500,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

out0_1 = F(Empty, Empty, Empty)

---

**Adding features in this order**

<table>
<thead>
<tr>
<th>Feature Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 2</td>
</tr>
<tr>
<td>2 1 0 2</td>
</tr>
<tr>
<td>3 2 0 1</td>
</tr>
<tr>
<td>4 0 2 1</td>
</tr>
<tr>
<td>5 2 0 0</td>
</tr>
<tr>
<td>6 0 2 0</td>
</tr>
</tbody>
</table>
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
<td>$500,000</td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

- out0_1 = F(Empty, Empty, Empty)
- out1_1 = F(40 years, Empty, Empty)
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Age</strong></td>
<td>40 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>#Rooms</strong></td>
<td>3 rooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>SF Downtown</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>$500.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

\[
\begin{align*}
\text{out0}_1 &= F(\text{Empty, Empty, Empty}) \\
\text{out1}_1 &= F(40 \text{ years, Empty, Empty}) \\
\text{out2}_1 &= F(40 \text{ years, 3 rooms, Empty})
\end{align*}
\]
**SHAPLEY VALUES**

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td>#Rooms</td>
<td>3 rooms</td>
</tr>
<tr>
<td>Price</td>
<td>$500,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

- out0_1 = F(Empty, Empty, Empty)
- out1_1 = F(40 years, Empty, Empty)
- out2_1 = F(40 years, 3 rooms, Empty)
- out3_1 = F(40 years, 3 rooms, SF Downtown)
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
<td>$500.000</td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction?

- out0_1 = F(Empty, Empty, Empty)
- out1_1 = F(40 years, Empty, Empty)
- out2_1 = F(40 years, 3 rooms, Empty)
- out3_1 = F(40 years, 3 rooms, SF Downtown)

---

#### Feature Permutations

<table>
<thead>
<tr>
<th>Feature Permutations</th>
<th>Adding features in this order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td></td>
</tr>
<tr>
<td>1 0 2</td>
<td></td>
</tr>
<tr>
<td>2 0 1</td>
<td></td>
</tr>
<tr>
<td>3 2 0</td>
<td></td>
</tr>
<tr>
<td>4 0 2</td>
<td></td>
</tr>
<tr>
<td>5 2 0</td>
<td></td>
</tr>
<tr>
<td>6 0 2</td>
<td></td>
</tr>
</tbody>
</table>

#### Feature Contributions for the 1st permutation

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>out1_1 - out0_1</td>
<td>out2_1 - out1_1</td>
<td>out3_1 - out2_1</td>
<td></td>
</tr>
</tbody>
</table>
What are the contributions of each feature for this prediction?

F(40 years, 3 rooms, SF Downtown) = $500,000
**SHAPLEY VALUES**

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
<td>$500,000</td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

\[
\text{out0\_2} = F(\text{Empty, Empty, Empty})
\]
## SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>#Rooms</td>
<td>3 rooms</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Location</td>
<td>SF Downtown</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Price</td>
<td>$500.000</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction?

out0_2 = F(Empty, Empty, Empty)

out1_2 = F(Empty, 3 rooms, Empty)
What are the contributions of each feature for this prediction?

\[
\text{out0}_2 = F(\text{Empty, Empty, Empty}) \\
\text{out1}_2 = F(\text{Empty, 3 rooms, Empty}) \\
\text{out2}_2 = F(40 \text{ years, 3 rooms, Empty})
\]
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>Feature Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 2</td>
</tr>
<tr>
<td>1</td>
<td>0 1 2</td>
</tr>
<tr>
<td>2</td>
<td>1 0 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
<td>$500.000</td>
</tr>
</tbody>
</table>

\[ F(40 \text{ years}, 3 \text{ rooms}, \text{SF Downtown}) = \$500.000 \]

What are the contributions of each feature for this prediction?

- out0_2 = F(Empty, Empty, Empty)
- out1_2 = F(Empty, 3 rooms, Empty)
- out2_2 = F(40 years, 3 rooms, Empty)
- out3_2 = F(40 years, 3 rooms, SF Downtown)

Adding features in this order:

\[ F(40 \text{ years}, 3 \text{ rooms}, \text{SF Downtown}) = \$500.000 \]
### SHAPLEY VALUES

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td></td>
</tr>
<tr>
<td>#Rooms</td>
<td>3 rooms</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>SF Downtown</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>$500,000</td>
<td></td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

\[
\text{out0}_2 = F(\text{Empty}, \text{Empty}, \text{Empty}) \\
\text{out1}_2 = F(\text{Empty}, 3\text{ rooms}, \text{Empty}) \\
\text{out2}_2 = F(40\text{ years}, 3\text{ rooms}, \text{Empty}) \\
\text{out3}_2 = F(40\text{ years}, 3\text{ rooms}, \text{SF Downtown})
\]

### Feature Permutations

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Adding features in this order:

- F(40 years, 3 rooms, SF Downtown) = $500,000

### Feature Contributions for the 2nd permutation

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 years</td>
<td>3 rooms</td>
<td>SF Downtown</td>
</tr>
<tr>
<td>out2_2 - out1_2</td>
<td>out1_2 - out0_2</td>
<td>out3_2 - out2_2</td>
</tr>
</tbody>
</table>
**SHAPLEY VALUES**

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Rooms</td>
<td>3 rooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>SF Downtown</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>$500,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

<table>
<thead>
<tr>
<th>Age</th>
<th>#Rooms</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>out1_1 - out0_1</td>
<td>out2_1 - out1_1</td>
<td>out3_1 - out2_1</td>
</tr>
<tr>
<td>out2_2 - out1_2</td>
<td>out1_2 - out0_2</td>
<td>out3_2 - out2_2</td>
</tr>
</tbody>
</table>

**Feature Permutations**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
**SHAPLEY VALUES**

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>40 years</td>
<td>#Rooms</td>
<td>3 rooms</td>
</tr>
<tr>
<td>Location</td>
<td>SF Downtown</td>
<td>Price</td>
<td>$500.000</td>
</tr>
</tbody>
</table>

F(40 years, 3 rooms, SF Downtown) = $500,000

What are the contributions of each feature for this prediction?

<table>
<thead>
<tr>
<th>Feature IDs</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>out1_1 - out0_1</td>
<td>#Rooms</td>
<td>out2_1 - out1_1</td>
</tr>
<tr>
<td>Location</td>
<td>out2_2 - out1_2</td>
<td>Price</td>
<td>out3_1 - out2_1</td>
</tr>
<tr>
<td>#Rooms</td>
<td>...</td>
<td>Location</td>
<td>out3_2 - out2_2</td>
</tr>
</tbody>
</table>

Sum and normalize by 6 the number of permutations
Formally Shapley values is defined as

$$\Phi(x_i) = \sum_{S \subseteq N \setminus i} \frac{S!(N - S - 1)!}{N!} [F_{S \cup \{i\}}(x_{S \cup \{i\}}) - F_S(x_S)],$$

where $N \in \mathbb{Z}^M$ is the total number of features and $S \in N$ is a subset of features.
SHAPLEY VALUES

• Formally Shapley values is defined as

\[
\Phi(x_i) = \sum_{S \subseteq N \setminus i} \frac{S!(N - S - 1)!}{N!} \left[ F_{S \cup \{i\}}(x_{S \cup \{i\}}) - F_S(x_S) \right],
\]

where \( N \in \mathbb{Z}^M \) is the total number of features and \( S \subseteq N \) is a subset of features.

Limitations

• Evaluates exponential number of feature perturbations

  • To mitigate this issue we approximate Shapley values using only a limited number of feature perturbations (Shapely Value Sampling, Polynomial calculation of the Shapley value based on sampling, Castro, et. at., 2009)
AXIOMS OF SHAPLEY VALUES

- **Symmetry**

  - For any function $F$ and all $S_{i,j}$, if features $i$ and $j$ are interchangeable then
  
  $F(S \cup \{i\}) = F(S \cup \{j\})$
AXIOMS OF SHAPLEY VALUES

- **Symmetry**
  - For any function $F$ and all $S_{i,j}$, if features $i$ and $j$ are interchangeable then
  \[ F(S \cup \{i\}) = F(S \cup \{j\}) \]

- **Dummy**
  - For any function $F$ and all $S_i$ if $i$ is a dummy feature, $F(S \cup \{i\}) = F(S)$
AXIOMS OF SHAPLEY VALUES

• **Symmetry**
  
  - For any function $F$ and all $S_{i, j}$, if features $i$ and $j$ are interchangeable then
  
  $$F(S \cup \{i\}) = F(S \cup \{j\})$$

• **Dummy**
  
  - For any function $F$ and all $S_{i}$ if $i$ is a dummy feature, $F(S \cup \{i\}) = F(S)$

• **Additivity**
  
  - The attribution of the linear combination of two functions $F_1$ and $F_2$ is equal to the linear combination of attributions for each of two functions
SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

- Approximates Shapley Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is an interpretable representation of input $x$, $M$ is the number of interpretable features
SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is a interpretable representation of input $x$, $M$ is the number of interpretable features
- $g(x')$ a interpretable explanation model
SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]

- $x' \in \{0,1\}^M$ is a interpretable representation of input $x$, $M$ is the number of interpretable features

- $g(x')$ a interpretable explanation model

- $x = h_x(x')$ - transforming interpretable input into original version
SHAPLEY ADDITITIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]

- \( x' \in \{0, 1\}^M \) is a interpretable representation of input \( x \), \( M \) is the number of interpretable features

- \( g(x') \) a interpretable explanation model

- \( x = h_x(x') \) - transforming interpretable input into original version

- \( f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \) - additivity property of explanations
Let’s assume that

- \( S \) is the set of non-zero indices in \( z' \) and \( z_S \) has missing values for features that are not in \( S \)
- \( \bar{S} \) is a set of features that are not in \( S \) and \( z_{\bar{S}} \) has missing values that are not in \( \bar{S} \)
SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

- Let’s assume that
  - $S$ is the set of non-zero indices in $z'$ and $z_S$ has missing values for features that are not in $S$.
  - $\bar{S}$ is a set of features that are not in $S$ and $z_{\bar{S}}$ has missing values that are not in $\bar{S}$.
- Approximating with conditional expectation
  - $f_{\hat{x}}(z) = f(h_{\hat{x}}(z')) = E[f(z) | z_S]$  
    SHAP explanation model interpretable for input representation
  - $\approx f([z_s, E[z_{\bar{s}}]])$  
    Assume interpretable model linearity
SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

- Approximating similar to Lime

\[
L(f, g, \pi) = \sum_{z \in Z, z' \in Z'} \pi_x'(z') (f(z) - g(z'))^2
\]

\[
\pi_x'(z') = \frac{(M - 1)}{(M \choose |z'|)|z'|((M - |z'|)}
\]

Similarity metric
PRACTICAL APPLICATIONS OF ATTRIBUTION ALGORITHMS
A MODEL INTERPRETABILITY LIBRARY FOR PYTORCH

MULTIMODAL

EXTENSIBLE

EASY TO USE

class MyAttribution(Attribution):
    def attribute(self, input, ...):
        attributions = self._compute_attr(input, ...)
        # <Add any logic necessary for attribution>
        return attributions

visualize_image_attr(attr_algo.attribute(input), ...)

WHAT DOES THE CAPTUM LIBRARY OFFER?

A number of gradient and perturbation-based attribution algorithms to interpret:
WHAT DOES THE CAPTUM LIBRARY OFFER?

A number of gradient and perturbation-based attribution algorithms to interpret:

- **Primary Attribution** -> Output predictions with respect to inputs
- Layer Attribution -> Output predictions with respect to all neurons in the layers
- Neuron Attribution -> Neurons with respect to inputs
A number of gradient and perturbation-based attribution algorithms to interpret:

- **Primary Attribution** -> Output predictions with respect to inputs
- **Layer Attribution** -> Output predictions with respect to all neurons in the layers
- **Neuron Attribution** -> Neurons with respect to inputs
A number of gradient and perturbation-based attribution algorithms to interpret:

- **Primary Attribution** -> Output predictions with respect to inputs
- **Layer Attribution** -> Output predictions with respect to all neurons in the layers
- **Neuron Attribution** -> Neurons with respect to inputs
ATTRIBUTION ALGORITHMS

Attribute model output (or internal neurons) to input features

- SHAP Methods
  - GradientSHAP
  - DeepLiftSHAP
  - KernelSHAP
  - Input * Gradient
  - GuidedGradCam
- Integrated Gradients
- Saliency
- DeepLift
- Shapely Value Sampling
- FeatureAblation / FeaturePermutation
- Occlusion
- LIME
- GuidedBackprop / Deconvolution

Attribute model output to the layers of the model

- SHAP Methods
  - LayerGradientSHAP
  - LayerDeepLiftSHAP
- InternalInfluence
  - GradCam
- LayerActivation
- LayerDeepLift
- LayerFeatureAblation
- LayerIntegratedGradients
- LayerGradientXAActivation
- LayerConductance

NoiseTunnel (Smoothgrad, Vargrad, Smoothgrad Square)
PRACTICAL APPLICATIONS WITH CAPTUM
INTERPRETING THE PREDICTIONS OF AN IMAGE CLASSIFICATION MODEL

BBX Model

Trained on CIFAR DataSet

Plane
from captum.attr import Saliency
from captum.attr import visualization as viz

# Creating and instance of Saliency algorithm
attr_algo = Saliency(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                           target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attr,
                         original_image,
                         method="blended_heat_map",
                         sign="absolute_value",
                         show_colorbar=True,
                         title="Overlayed Gradient Magnitudes")
from captum.attr import Saliency
from captum.attr import visualization as viz

# Creating and instance of Saliency algorithm
attr_algo = Saliency(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attr,
                          original_image,
                          method="blended_heat_map",
                          sign="absolute_value",
                          show_colorbar=True,
                          title="Overlayed Gradient Magnitudes"
from captum.attr import IntegratedGradients
from captum.attr import visualization as viz

# Creating an instance of Integrated Gradients algorithm
attr_algo = IntegratedGradients(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlayed Integrated Gradients Magnitudes")
from captum.attr import IntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Integrated Gradients algorithm
attr_algo = IntegratedGradients(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                           target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated Gradients Magnitudes")
from captum.attr import LayerIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Layer Integrated Gradients algorithm
attr_algo = LayerIntegratedGradients(cifar_net, cifar_net.conv1)

# Computing the attributions for plane w.r.t. conv1 layer output
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# interpolate layer output in order to match input size
attrs = LayerAttribution.interpolate(attrs, (32,32))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlayed Integrated Gradients Magnitudes"
from captum.attr import LayerIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Layer Integrated Gradients algorithm
attr_algo = LayerIntegratedGradients(cifar_net, cifar_net.conv1)

# Computing the attributions for plane w.r.t. conv1 layer output
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Interpolate layer output in order to match input size
attrs = LayerAttribution.interpolate(attrs, (32,32))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlaid Integrated Gradients Magnitudes")
from captum.attr import NeuronIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Neuron Integrated Gradients algorithm
attr_algo = NeuronIntegratedGradients(cifar_net, cifar_net.conv1)

# Computing the attributions for the sum of all neurons in the output of conv1 layer w.r.t. inputs
attrs = attr_algo.attribute(image,
    neuron_selector = lambda x: x.sum(axis=(0,1,2))
)

# Visualizing attributions
viz.visualize_image_attr(attrs,
    original_image,
    method="Blended_heat_map",
    sign="all",
    show_colorbar=True,
    title="Overlayed Integrated Gradients Magnitudes"
from captum.attr import NeuronIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Neuron Integrated Gradients algorithm
attr_algo = NeuronIntegratedGradients(cifar_net, cifar_net.conv1)

# Computing the attributions for the sum of all neurons in the output of conv1 layer w.r.t. inputs
attrs = attr_algo.attribute(image,
                            neuron_selector = lambda x: x.sum(axis=(0,1,2))
)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="Blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlayed Integrated Gradients Magnitudes";
)
from captum.attr import ShapleyValueSampling
from captum.attr import visualization as viz

# Creating and instance of Shapley Value Sampling algorithm
attr_algo = ShapleyValueSampling(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attr = attr_algo.attribute(image,
                          target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlayed Shapley Value Sampling")
from captum.attr import ShapleyValueSampling
from captum.attr import visualization as viz

# Creating and instance of Shapley Value Sampling algorithm
attr_algo = ShapleyValueSampling(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                             target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                          original_image,
                          method="blended_heat_map",
                          sign="all",
                          show_colorbar=True,
                          title="Overlayed Shapley Value Sampling")
attributions = Attribution(forward_func, ...).attribute(input, ...)
USING CAPTUM FOR BRAIN MRI SEGMENTATION
U-NET MODEL FOR BRAIN MRI ABNORMALITY SEGMENTATION

Model Link: https://pytorch.org/hub/mateuszbuda_brain-segmentation-pytorch_unet/
Dataset Link: https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation
U-NET MODEL FOR BRAIN MRI ABNORMALITY SEGMENTATION
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda'\
    '/brain-segmentation-pytorch',
    'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)
```
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' '/brain-segmentation-pytorch',
    'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define custom forward function for computing the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
    internal_batch_size=1)
```
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment
- Visualize Attributions

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' /
    '/brain-segmentation-pytorch',
    'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
    internal_batch_size=1)

from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_ig,
    orig_img_test,
    methods=["heat_map"],
    signs="positive",
    ... )
```
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' /'brain-segmentation-pytorch',
            'unet',
            in_channels=3, out_channels=1,
            init_features=32, pretrained=True)

# define custom forward function for computing the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
                        internal_batch_size=1)

from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_ig,
                                   orig_img_test,
                                   methods=['heat_map'],
                                   signs=['positive'],
                                   ... )
```
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' /
    '/brain-segmentation-pytorch',
    'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_gbp = gbp.attribute(inp_img_tensor)
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \ '/brain-segmentation-pytorch', 'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_gbp = gbp.attribute(inp_img_tensor)

from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_gbp,
    orig_img_test,
    methods=['heat_map'],
    signs=['positive'],
    ... )
• Loading U-Net model trained on Brain MRI images
• Summarizing the output of the prediction using custom forward function
• Compute Guided Back Prop for the prediction segment

import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
'/brain-segmentation-pytorch', 'unet', 
    in_channels=3, out_channels=1, 
    init_features=32, pretrained=True)

# define custom forward function for computing # the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)

from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise # mean and std
attr_gbp = gbp.attribute(inp_img_tensor)

from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_gbp, orig_img_test, methods=['heat_map'], signs=['positive'], ... )
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda'\
    '/brain-segmentation-pytorch',
    'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define a wrapper model for computing layer attributions

class MyCustomModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = unet_model

    def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)
Loading U-Net model trained on Brain MRI images

Wrapping unet model with a wrapper model and returning the sum of output predictions

Compute Layer Grad Cam for a specific layer `upconv4`

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' '/brain-segmentation-pytorch',
'unet',
    in_channels=3, out_channels=1,
    init_features=32, pretrained=True)

# define a wrapper model for computing # layer attributions

class MyCustomModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = unet_model

    def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)

from captum.attr import LayerGradCam

my_model = MyCustomModule()

# creating an instance of Layer GradCam
lgc = LayerGradCam(my_model, my_model.model.upconv4)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_lgc = lgc.attribute(inp_img_tensor)
```
**Layer GradCam for MRI Abnormality Segmentation**

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Layer Grad Cam for a specific layer `upconv4`
- Interpolate attributions to the input size

```python
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
    '/brain-segmentation-pytorch', 
    'unet',
    in_channels=3, out_channels=1, 
    init_features=32, pretrained=True)

# define a wrapper model for computing # layer attributions
class MyCustomModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = unet_model

    def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)

from captum.attr import LayerGradCam

my_model = MyCustomModule()

# creating an instance of Layer GradCam
lgc = LayerGradCam(my_model, my_model.model.upconv4)

# inp_img_tensor is normalized using channel-wise # mean and std
attr_lgc = lgc.attribute(inp_img_tensor)

# interpolate attributions to match input size
attr_lgc_inter = LayerAttribution.
    interpolate(attr_lgc, .... )
```
Visualizing attribution scores for layer `upconv4`
Visualizing attribution scores for layer `upconv4`
EVALUATION OF MODEL INTERPRETABILITY
EVALUATION OF MODEL INTERPRETABILITY

- No clear guidance on how to quantify the quality of interpretations / explanations
- Quantitative metrics are often domain specific
- Visual evaluation can be misleading or seen as a confirmation bias
A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)
A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)

- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement
A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)

- More strict evaluation than functional grounded one
- Human evaluation is required but not by domain experts
- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement
A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)

- Validate explanations in products by end users
- Requires domain experts, good definition of how to evaluate the quality in an unbiased manner
- More strict evaluation than functional grounded one
- Human evaluation is required but not by domain experts
- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement
EVALUATION METRICS > SENSITIVITY - MAX

- Measures the sensitivity of explanations to subtle input perturbations using Monte-Carlo sampling-based approximation ([On the (In)fidelity and Sensitivity of Explanations, Yeh, et. al., 2019](#))

- Given input $x \in \mathbb{R}^N$, perturbed input $y \in \mathbb{R}^N$, perturbation radius $r \in \mathbb{R}$, a NN function $F : \mathbb{R}^N \rightarrow \mathbb{R}$ and an explanation function $\Phi : F \times \mathbb{R}^N \rightarrow \mathbb{R}^N$

$$SENS_{MAX}(\Phi, F, x, r) = \max_{\|y-x\| \leq r} \frac{||\Phi(F, y) - \Phi(F, x)||}{||\Phi(F, x)||}$$
EVALUATION METRICS > INFIDELITY

- Measures mean-squared error between dot product of input perturbation and explanation and differences between the predictor function at its input and perturbed input (*On the (In)fidelity and Sensitivity of Explanations, Yeh, et. al., 2019*)

- Completeness property is a special case of infidelity metric

Given input $x \in \mathbb{R}^N$, a NN function $F : \mathbb{R}^N \rightarrow \mathbb{R}$, a meaningful perturbation $I \in \mathbb{R}^N$ with a probability measure $\mu_I$

\[
\text{INF}_{\mu_I}(\Phi, F, x) = \mathbb{E}_{I \sim \mu_I}[(I^T \Phi(F, x) - (F(x) - F(x - I)))^2]
\]
CONCEPT-BASED MODEL
INTERPRETABILITY
CONCEPT-BASED MODEL INTERPRETABILITY

- Explaining model predictions on the basis of pre-defined concepts

Stripes

Random
CONCEPT-BASED MODEL INTERPRETABILITY

- Explaining model predictions on the basis of pre-defined concepts

- Measures prediction sensitivity to high-level concepts
CONCEPT-BASED MODEL INTERPRETABILITY
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

• Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation

\[ f_L : \mathbb{R}^N \rightarrow \mathbb{R}^M \]

\[ h_{L,K} : \mathbb{R}^M \rightarrow \mathbb{R}^P \]
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation

Two conceptual vectors $f_L$ and $h_{L,K}$ are used to perform testing with concept activation vectors (TCAV).
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation

CAV is the vector orthogonal to the hyperplane of concept linear classifier

$f_L: \mathbb{R}^N \rightarrow \mathbb{R}^M$

$h_{L,K}: \mathbb{R}^M \rightarrow \mathbb{R}^P$
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

\[
\text{Sens}_{L,ZEBRA}^ZEBRA(f_L) = \frac{\partial h_{L,ZEBRA}(f_L)}{\partial f_L}(ZEBRA, TARGET K)
\]
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

\[
CAV_{SENS}^{STRIpes}_{L,Zebra}(x) = CAV_{STRIpes}^{STRIpes} \cdot Sens_{L}^{Zebra}(x)
\]
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

\[
TCAV_{STRIpes,LEBRA}(x) = CAV_{STRIpes,LEBRA}(x) \\
CAV_{SENS_{STRIpes,LEBRA}}(x) \geq 0
\]
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

**In a general case**

\[
TCAV_{L,CLASS}^{CONCEPT}(inputs_{CLASS}) = \frac{|CAV_{SENS_{L,CLASS}}^{CONCEPT}(inputs_{CLASS}) > 0|}{|inputs_{CLASS}|}
\]
TCAV > STATISTICAL SIGNIFICANCE TESTING

- TCAV can potentially learn meaningless CAVs for a meaningful concept
- A CAV generated for a random concept can potentially be meaningful
TCAV > STATISTICAL SIGNIFICANCE TESTING

- TCAV can potentially learn meaningless CAVs for a meaningful concept
- A CAV generated for a random concept can potentially be meaningful
- Steps we can take to mitigate those issues
  - Two sided statistical significance tests
    - Against large number of random concepts
    - A meaningful concept will stand out with high TCAV score among most random concepts
TCAV > LIMITATIONS

- Concepts has to be pre-defined in advance
  - Time consuming process
- Learning meaningless CAVs
  - Statistical significance testing for multiple random concepts
  - Computationally time and memory intensive
Automatic Concept Extraction (ACE) for images (Towards Automatic Concept-based Explanations, Ghorbani et al., 2019)

Identifying a sufficient set of concepts that describe our prediction, ConceptSHAP (On Completeness-aware Concept-Based Explanations in Deep Neural Networks, Yeh, et al., 2020)
MODEL COMPARISON
**PRINCIPLE COMPONENT ANALYSIS (PCA)**

- Visualizing high dimensional embedding spaces
- Principal Component Analysis (PCA)
  
  (Lill. On lines and planes of closest fit to systems of points in space, Pearson F.R.S, 1901)
PRINCIPLE COMPONENT ANALYSIS (PCA)

- Visualizing high dimensional embedding spaces
- Principal Component Analysis (PCA) (Lill. On lines and planes of closest fit to systems of points in space, Pearson F.R.S, 1901)
- Projects high dimensional layer embedding vectors to a lower dimensional space that captures maximum variance in the data
PRINCIPLE COMPONENT ANALYSIS (PCA)

- Two Principle components for CIFAR-10 test dataset
- Colored dots are examples from test dataset

Source: Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019
• Comparing embedding representations for a pair of layers in a model or across multiple models

$$f_L : R^N \rightarrow R^M$$  $$f_L : R^N \rightarrow R^M$$
CANONICAL CORRELATION ANALYSIS (CCA)

- Comparing embedding representations for a pair of layers in a model or across multiple models
- **Canonical Correlation Analysis (CCA)** (Relations between two sets of variates, Hotelling, 1936)
  Explains correlation between embedding representations of a pair of layers
CANONICAL CORRELATION ANALYSIS (CCA)

- Comparing embedding representations for a pair of layers in a model or across multiple models
- **Canonical Correlation Analysis (CCA)** *(Relations between two sets of variates, Hotelling, 1936)*
  
  Explains correlation between embedding representations of a pair of layers

\[
f_L : R^N \rightarrow R^M \quad f_L : R^N \rightarrow R^M
\]
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** *(SVCCA, Raghu, et. al., 2017)*
  Applies CCA on the top directions of both layers that capture the most variance
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** (SVCCA, Raghu, et al., 2017)
  
  Applies CCA on the top directions of both layers that capture the most variance

\[
SVD(f_{l2}(x)) : R^m \rightarrow R^d \quad \text{SVD}(f_{l6}(x)) : R^m \rightarrow R^d
\]
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** ([SVCCA, Raghu, et. al., 2017](#))

  Applies CCA on the top directions of both layers that capture the most variance.

\[
SVD(f_{l2}(x)) : R^m \rightarrow R^d \\
\text{CCA}(\text{EigenVec}_{l2}, \text{EigenVec}_{l6})
\]

\[
SVD(f_{l6}(x)) : R^m \rightarrow R^d
\]
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** ([SVCCA, Raghu, et. al., 2017](#))
  
  Applies CCA on the top directions of both layers that capture the most variance.

\[
SVD(f_L(x)) : R^m \rightarrow R^d \quad SVD(f_L(x)) : R^m \rightarrow R^d
\]

\[
CCA(EigenVec_{l2}, EigenVec_{l6})
\]

\[
CORR(CCA_{EigenVec_{l2}}, CCA_{EigenVec_{l6}})
\]
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

• Comparing the layers of two pre-trained CIFAR-10 models that used different model initialization

Source: SVCCA, Raghu, et. al., 2017
SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- Layer similarities between trained and during different stages of training for CIFAR-10 model

Source: SVCCA, Raghu, et. al., 2017
PROJECTED WEIGHT CANONICAL CORRELATION ANALYSIS (PWCCA)

- An improvement of SVCCA that is better at distinguishing noise from important signal

- **Projected Weight Canonical Correlation Analysis (PWCCA)** *(Insights on representational similarity in neural networks with canonical correlation, Marcos, et. al., 2018)*

Weights CCA vectors based on how much the original input vector accounts for the CCA canonical covariates.
CENTERED KERNEL ALIGNMENT (CKA)

- Compares embedding representations of layer pairs in a model or across multiple models

- **Centered Kernel Alignment (CKA)** *(Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019)*
  A generalization of dot product similarity metric by Kernel Hilbert Spaces
CENTERED KERNEL ALIGNMENT (CKA)

- Compares embedding representations of layer pairs in a model or across multiple models

- **Centered Kernel Alignment (CKA)** *(Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019)*
  
  A generalization of dot product similarity metric by Kernel Hilbert Spaces

Given $X = F_{L2}(input)$, $Y = F_{L6}(input)$

$$\langle vec(XX^T), vec(YY^T) \rangle = tr(XX^TYY^T) = ||Y^T X||_F^2 \quad (1)$$
CENTERED KERNEL ALIGNMENT (CKA)

• Given $X = F_{L2}(\text{input})$, $Y = F_{L6}(\text{input})$

$$
\langle \text{vec}(XX^T), \text{vec}(YY^T) \rangle = \text{tr}(XX^TYY^T) = \| Y^T X \|_F^2 \quad (1)
$$

can be generalized with Hilbert-Schmidt Independence Criterion for centered $X$ and $Y$

Linear or RBF kernels: $K_{ij} = k(x_i, x_j), L_{ij} = l(y_i, y_j)$

$$
\text{HSIC}(K, L) = \frac{1}{(n - 1)^2} \text{tr}(KHLH), \quad H \text{ is centering matrix}
$$
 CENTERED KERNEL ALIGNMENT (CKA)

- Given $X = F_{L2}(input)$, $Y = F_{L6}(input)$

$$\langle vec(XX^T), vec(YY^T) \rangle = tr(XX^T YY^T) = \|Y^T X\|_F^2$$

can be generalized with Hilbert-Schmidt Independence Criterion for centered $X$ and $Y$

Linear or RBF kernels: $K_{ij} = k(x_i, x_j), L_{ij} = l(y_i, y_j)$

$$HSIC(K, L) = \frac{1}{(n-1)^2} tr(KHLH) , \enspace H \text{ is centering matrix}$$

In order to be invariant to isotopic scaling

$$CKA(K, L) = \frac{HSIC(K, L)}{\sqrt{HSIC(H, H)HSIC(L, L)}}$$
CENTERED KERNEL ALIGNMENT (CKA)

- CKA using linear and RBF Kernels for CIFAR-10 model trained with two different parameter initializations

Source: Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019
CENTERED KERNEL ALIGNMENT (CKA)

- CCA vs SVCCA vs CKA for CIFAR-10 model trained with two different parameter initializations.

Source: *Similarity of Neural Network Representations Revisited*, Kornblith, et. al., 2019
SUMMARY

- Five desiderata of model interpretability research
- Black Box vs inherently interpretable models
- Explaining with gradient and perturbation-based attribution algorithms
- Attribution algorithms for image classification and segmentation
- Evaluating the quality of model explanations
- Concept-based model interpretability
- Model comparison and correlation analysis
OTHER DIRECTIONS OF MODEL INTERPRETABILITY RESEARCH

• Optimization-based visualizations

  Identifying concepts learned by a neuron or groups of neurons

• Adversarial robustness and model interpretability
  • More robust models and more interpretable gradients