Propagation-based explanations

CSEP 590B: Explainable AI
Ian Covert & Su-In Lee
University of Washington
Previously

- Feature importance explanations
  - Removal-based explanations
  - Shapley values

- Today: propagation-based explanations
Today

- Section 1
  - Backprop review
  - Gradient-based explanations
- Section 2
  - Modified backprop variants
  - Propagation vs. removal-based explanations
Setup

- Consider a classification model $f(x)$
- $f_y(x)$ is probability for class $y$
- Input is $x \in \mathbb{R}^d$ (must be continuous)
- Assume $f$ is differentiable (a neural network)
Review: backpropagation

- Deep learning models are trained using stochastic gradient descent (SGD)
  - Get a minibatch of examples
  - Calculate predictions and loss
  - Calculate gradients using “backprop” algorithm
Backpropagation

Input layer

\( x \)

\( h_1 \)

\( h_2 \)

\( \hat{y} \)

Output layer

Network weights
Backpropagation

Network parameters
\[ \theta = ([w_1, b_1], [w_2, b_2], [w_{out}, b_{out}]) \]

Forward pass
\[ h_1 = \sigma(w_1 x + b_1) \]
\[ h_2 = \sigma(w_2 h_1 + b_2) \]
\[ \hat{y} = w_{out} h_2 + b_{out} \]

Gradient descent
\[ \theta_{t+1} = \theta_t - \eta \nabla_{\theta} \mathcal{L}(\theta_t) \]
Backpropagation (cont.)

- Model loss is mean prediction error:

$$
\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x^i; \theta), y^i)
$$

- Gradient calculation:

$$
\nabla_\theta \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(f(x^i; \theta), y^i)
$$
Chain rule

- Calculate gradients for all parameters $\theta$ using chain rule
  - Get gradients for last hidden layer
  - Then for the previous layer
  - Then the layer before that...

- Backpropagation = propagating gradients backward through the network

Rumelhart et al., “Learning representations by back-propagating errors” (1986)
Chain rule (cont.)

\[ \ell(\theta) = \sum \ell(\hat{y}, y) \]
Propagation-based explanations

- Use backprop idea to quantify feature importance

- Rather than gradients w.r.t. parameters, calculate gradients w.r.t. inputs
Input gradients

\[ \ell(\theta) = \sum \ell(\hat{y}, y) \]

\[ \frac{\partial L}{\partial x} \quad \frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \frac{\partial L}{\partial w_{\text{out}}} \]

\[ \frac{\partial L}{\partial h_1} \quad \frac{\partial L}{\partial h_2} \]

\[ \frac{\partial L}{\partial b_1} \quad \frac{\partial L}{\partial b_2} \quad \frac{\partial L}{\partial b_{\text{out}}} \]
Input gradients

Gradient of prediction (instead of loss)

\[
\frac{\partial f_y}{\partial x}, \quad \frac{\partial f_y}{\partial w_1}, \quad \frac{\partial f_y}{\partial w_2}, \quad \frac{\partial f_y}{\partial w_{\text{out}}}, \quad \frac{\partial f_y}{\partial b_1}, \quad \frac{\partial f_y}{\partial b_2}, \quad \frac{\partial f_y}{\partial b_{\text{out}}}
\]

\( f_y(x; \theta) = \hat{y} \)
Intuition

- Partial derivatives represent sensitivity to small perturbations

- Mathematically:

\[
\frac{\partial f_y}{\partial x_i}(x) = \lim_{\epsilon \to 0} \frac{f_y(x + e_i \cdot \epsilon) - f_y(x)}{\epsilon}
\]

Delta from small change in \(i\)th direction

Limit as change becomes very small

Measured relative to the size of change
Today

- Section 1
  - Backprop review
  - Gradient-based explanations

- Section 2
  - Modified backprop variants
  - Propagation vs. removal-based explanations
Application to XAI

- **Idea:** find features that cause large output changes when perturbed

- **Remark:** quantifies feature sensitivity, but not necessarily related to feature removal
Vanilla gradients

- For an input $x$ and label $y$, calculate gradient of the prediction $f_y(x)$:

$$a_i = \frac{\partial f_y}{\partial x_i}(x)$$

- Can optionally use absolute value:

$$a_i = \left| \frac{\partial f_y}{\partial x_i}(x) \right|$$

Vanilla gradients (cont.)
Variation 1: SmoothGrad

- Average gradients across inputs near \( x \)
- E.g., add Gaussian noise:

\[
a_i = \mathbb{E} \left[ \frac{\partial f_y}{\partial x_i} (x + \epsilon) \right] \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)
\]

- In practice, use small number of \( \epsilon \) samples (50)
- Must tune \( \sigma^2 \) to an appropriate level

Smilkov et al., “SmoothGrad: Removing noise by adding noise” (2017)
SmoothGrad (cont.)

Standard saliency maps
Varying levels of input noise
Variation 2: Grad x Input

- Multiply gradient by input values:

\[ a_i = x_i \cdot \frac{\partial f_y}{\partial x_i}(x) \]

Shrikumar et al., “Not just a black box: Learning important features through propagating activation differences” (2016)
**Grad x Input (cont.)**

- **Interpretation:** consider the model’s first-order Taylor expansion around $x_0$

  \[ f_y(x) \approx f_y(x_0) + (x - x_0)^\top \frac{\partial f_y}{\partial x}(x_0) \]

- Gradient gives linearized version of model (like replacing a function with its tangent line)

- Grad x Input = approximates impact of setting the input to zero
  - Similar to occlusion (see previous lecture)
Variation 3: Integrated gradients

- Gradients can become “saturated”
- Model is sensitive to big input changes, but not small ones

In this region, $f(x)$ is insensitive to small $x$ changes

Saturation can yield small gradients, even for important inputs

IntGrad (cont.)

- **Idea:** address saturation issue by calculating gradients for rescaled images, $\alpha \cdot x$

$$
\frac{\partial f_y}{\partial x_i} (\alpha \cdot x) \text{ for } 0 \leq \alpha \leq 1
$$

- Integrate (average) gradients across range of rescaled images:

$$
\int_{\alpha=0}^{1} \frac{\partial f_y}{\partial x_i} (\alpha \cdot x) \, d\alpha
$$

- Multiply by the input feature value:

$$
a_i = x_i \int_{\alpha=0}^{1} \frac{\partial f_y}{\partial x_i} (\alpha \cdot x) \, d\alpha
$$
IntGrad (cont.)

- Implicitly relies on a zeros baseline
- Can instead use a non-zero baseline $x'$

\[
a_i = (x_i - x'_i) \int_{\alpha=0}^{1} \frac{\partial f_y}{\partial x_i} (x' + \alpha \cdot (x - x')) d\alpha
\]

- Related to a different idea from cooperative game theory: the Aumann-Shapley value
  - Different from previous Shapley value
  - Has its own axiomatic derivation (see Sundararajan et al.)
IntGrad (cont.)

- **Problem:** the integral is hard to calculate

- **Solution:** use Riemann sum approximation for $m$ regularly spaced values $\alpha_j \in [0,1]$: 

$$a_i \approx (x_i - x'_i) \frac{1}{m} \sum_{j=1}^{m} \frac{\partial f_y}{\partial x_i} (x' + \alpha_j \cdot (x - x'))$$
IntGrad (cont.)

Gradient with image rescaled by $\alpha$

Input image and trend of the pixel importance scores obtained from interior gradients.

Top label: reflex camera
Score: 0.993755
IntGrad (cont.)

Original image

Top label and score
Top label: reflex camera
Score: 0.993755

Top label: fireboat
Score: 0.999961

Top label: school bus
Score: 0.997033

Integrated gradients

Gradients at image
GradCAM

- In CNNs, hidden layers represent high-level visual concepts
- Hidden layers retain spatial information due to convolutional structure

Idea: explain models via the last convolutional layer instead of the input layer

CNN receptive fields

Receptive field grows at each layer, but remains localized
GradCAM procedure

- Denote final layer’s hidden representation as $A$
  - Size is $A \in \mathbb{R}^{w \times h \times c}$
  - Width $w$, height $h$, channels $c$
  - Each channel $k = 1, \ldots, c$ denoted as $A^k \in \mathbb{R}^{w \times h}$

- The final prediction $f_y(x)$ can be viewed as a function of representation $A$
  - E.g., $A \rightarrow$ global average pooling $\rightarrow$ MLP
GradCAM procedure (cont.)

$A$

$\hat{y}$

$f_y(x)$
GradCAM procedure (cont.)

- Calculate gradients w.r.t. $A$
  \[
  \frac{\partial f_y}{A_{ij}^k} \quad \text{for all } (i, j, k)
  \]

- Average gradients within each channel:
  \[
  \alpha_k^y = \frac{1}{wh} \sum_{ij} \frac{\partial f_y}{A_{ij}^k}
  \]

- Aggregate hidden representations across channel dimension using $\alpha_k^y$
  \[
  a_{ij} = \sum_{k=1}^c \alpha_k^y A_{ij}^k
  \]
GradCAM procedure (cont.)

- Often use thresholding function (suppress negative attributions):

\[ a_{ij} = \text{ReLU} \left( \sum_{k=1}^{c} \alpha_{k}^{y} A_{ij}^{k} \right) \]

- Can optionally upsample low-resolution scores \( a_{ij} \) to the original input size (e.g., bilinear upsampling)
GradCAM interpretation

- The values $\alpha^y_k$ represent smoothed or averaged gradient of class $y$ w.r.t. channel $k$.

- At each location, activations $A_{ij}^k$ are multiplied by averaged gradients and then aggregated.

- Similar to Grad x Input, but using a hidden layer instead of input layer.
  - Like a Taylor approximation of setting internal activations to zero.
GradCAM results

(a) Original Image  (b) Guided Backprop ‘Cat’  (c) Grad-CAM ‘Cat’  (d) Guided Grad-CAM ‘Cat’  (e) Occlusion map ‘Cat’  (f) ResNet Grad-CAM ‘Cat’

(g) Original Image  (h) Guided Backprop ‘Dog’  (i) Grad-CAM ‘Dog’  (j) Guided Grad-CAM ‘Dog’  (k) Occlusion map ‘Dog’  (l) ResNet Grad-CAM ‘Dog’
Other gradient-based methods

- **Guided backprop**: Springenberg et al., “Striving for simplicity: The all-convolutional net” (2014)
- **VarGrad**: Adebayo et al., “Local explanation methods for deep neural networks lack sensitivity to parameter values” (2018)
- **Expected gradients**: Erion et al., “Learning explainable models using attribution priors” (2020)
- **BlurIG**: Xu et al., “Attribution in scale and space” (2020)
Today

- Section 1
  - Backprop review
  - Gradient-based explanations
  - **10 min break**
- Section 2
  - Modified backprop variants
  - Propagation vs. removal-based explanations
Propagation-based explanations
(continued)

CSEP 590B: Explainable AI
Ian Covert & Su-In Lee
University of Washington
Today

- Section 1
  - Backprop review
  - Gradient-based explanations
- Section 2
  - Modified backprop variants
  - Propagation vs. removal-based explanations
Modified backpropagation

- Previous approaches rely on \textit{gradient} backprop.
- Others use heuristic backprop variants:
  - Calculate “importance” of internal nodes, propagate back to earlier ones.
  - Requires justification for different backprop heuristics.
Layer-wise relevance propagation (LRP)

- Intuition: iteratively calculate relevance scores for every layer of a model
  - Start with nodes in the last hidden layer
  - Move backwards through the model by splitting scores in the previous layer

LRP (cont.)

- Let $R^{(l+1)}_j \in \mathbb{R}$ denote relevance for $j$th node in layer $l + 1$
- Initialize $R^{(L)}_1 = f_y(x)$ for output node of interest
- Let $R^{(l,l+1)}_{i \leftarrow j}$ denote relevance message sent from $j$th node in layer $l + 1$ to $i$th node in layer $l$
- Want messages to satisfy two conservation properties

$$R^{(l+1)}_j = \sum_{i} R^{(l,l+1)}_{i \leftarrow j}$$  \hspace{2cm} \text{Summation of outgoing importance}

$$R^{(l)}_i = \sum_{j} R^{(l,l+1)}_{i \leftarrow j}$$  \hspace{2cm} \text{Summation of incoming importance}
The previous rules don’t define a unique procedure, so the authors propose multiple options.

For example, the “$\epsilon$-rule”

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

where $z_{ij} = w_{ij}^{(l+1)} h_i^{(l)}$, $z_j = \sum_i z_{ij} + b_j^{(l+1)}$, and $\epsilon > 0$

Finally, attributions are given by $a_i = R_i^{(1)}$ for $i = 1, \ldots, d$
LRP (cont.)
LRP discussion

- Arguably less intuitive than other methods, requires some heuristic choices (which “rule” to use?)
- Can be difficult to adapt to different architectures
  - E.g., does not automatically support residual connections (ResNet architecture), requires extension to transformers
Other modified backpropagation methods

- **DeepLIFT**: Shrikumar et al., “Not just a black box: Learning important features through propagating activation differences” (2016)
- **A unifying perspective**: Ancona et al., “Towards better understanding of gradient-based attribution methods” (2017)
- **Excitation backprop**: Zhang et al., “Top-down neural attention by excitation backprop” (2018)
Today

- Section 1
  - Backprop review
  - Gradient-based explanations

- Section 2
  - Modified backprop variants
  - Propagation vs. removal-based explanations
Many explanation methods

- Removal-based explanations
  - SHAP, LIME, RISE, Occlusion, permutation tests

- Propagation-based explanations
  - SmoothGrad, IntGrad, GradCAM

What should we use in practice?
Model flexibility

What kind of model are you explaining?

- Removal-based explanations are **model-agnostic**
  - Can work with any model class (DNNs, trees, etc.)

- Propagation-based explanations are mainly for neural networks
  - Usually require differentiation
  - Some even have architecture constraints
Data flexibility

What kind of data do you have?

- Removal-based explanations can handle discrete and continuous features
  - E.g., replace inputs with alternative values from the dataset

- Propagation-based methods only make sense for continuous features
  - Derivative = sensitivity to small change
  - Small changes are meaningless for discrete features
    - E.g., $x_i \in \{0, 1, 2\}$
Local or global

What kind of explanation do you need?

- Both removal- and propagation-based methods can produce local explanations

- Removal-based methods are better suited for global explanations
  - Can focus on a global model behavior (e.g., dataset loss)
  - To use a propagation-based method, we require an aggregation scheme (e.g., mean of local explanations)
Speed

Is speed important?

- Propagation-based methods are fast
  - Backward pass through DNN
  - Weak dependence on number of features

- Removal-based methods can be slow
  - Often require making predictions with many feature subsets
  - Shapley values are particularly challenging
Quality

Which explanation is most informative or correct?

- Theory can serve as a guide
  - E.g., Shapley value axioms, IntGrad axioms
- We can also take an empirical approach
  - Metrics for explanation quality (next lecture)

- **Perspective:** no explanation is *wrong*, but some procedures are misaligned with user questions
Popular methods

Which methods do most people use?

- A small number of methods dominate
- Depends on the data domain (tabular, image, NLP)
Tabular data

- **Permutation tests** are widely used for global feature importance

- **SHAP** is ubiquitous for local explanations
  - TreeSHAP is built into XGBoost, LGBM
  - KernelSHAP used for other models
Computer vision

- Gradient-based methods are currently most popular: GradCAM, IntGrad
  - Removal-based methods are usually too slow
  - Some papers try to fix this, but not popular (yet?)
    - Masking model: Dabkowski & Gal, “Real time image saliency for black box classifiers” (2017)
NLP

- NLP models (LSTMs, transformers) can use most methods
  - Gradient-based methods are popular
  - Removal-based explanations are slower, but leave-one-out (occlusion) is sometimes used

- For transformers, some use **attention** as explanation
  - Perhaps an *interpretable architecture*?
  - We’ll return to this in a later lecture
## Popular packages

<table>
<thead>
<tr>
<th>GitHub package</th>
<th>Description</th>
<th>Stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>slundberg/shap</td>
<td>SHAP variations (KernelSHAP, TreeSHAP, DeepSHAP, etc.)</td>
<td>15.8k</td>
</tr>
<tr>
<td>marcotcr/lime</td>
<td>LIME for images, tabular data</td>
<td>9.7k</td>
</tr>
<tr>
<td>utkuozbulak/pytorch-cnn-visualizations</td>
<td>Various gradient-based methods</td>
<td>6.4k</td>
</tr>
<tr>
<td>jacobgil/pytorch-grad-cam</td>
<td>GradCAM + GradCAM variations</td>
<td>4.5k</td>
</tr>
<tr>
<td>pytorch/captum</td>
<td>Various gradient-based methods + SHAP</td>
<td>3.0k</td>
</tr>
<tr>
<td>sicara/tf-explain</td>
<td>Various gradient-based methods + occlusion</td>
<td>906</td>
</tr>
<tr>
<td>kundajelab/deeplift</td>
<td>DeepLIFT</td>
<td>616</td>
</tr>
<tr>
<td>ankurtaly/Integrated-Gradients</td>
<td>IntGrad</td>
<td>453</td>
</tr>
</tbody>
</table>