SHAP

SHapley **A**dditive ex**P**lanations

Andrea Masi - Victor Badenas

May 2021

Summary

Previous Knowledge

Additive Feature Attribution Methods

Shapley Values

SHAP

Model Agnostic:

- Kernel SHAP

Model Dependant:

- Linear SHAP
- Tree SHAP
- Deep SHAP

References

Previous Knowledge

Additive Feature Attribution Methods

What does it means? if we decompose it...

- **Feature attribution:** Indicates how much a quantity of interest of our model rely on each feature. ”How much importance”
- **Additive** importance: Summing the importances should lead to a quantity that makes sense.

⇒ Additive Feature Attribution Methods

“The best explanation of a simple model is the model itself”

⇒ If you are able to understand it!

Additive Feature Attribution Methods

Instead, we must use a simpler explanation model:

- where ***g*** is the explanation model;
- $z' \in \{0, 1\}^M$ is the simplified feature vector;
- ***M*** is feature combinations;
- the importance **ϕ** of a given feature (**j**)

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

For instance, we have 3 features x = $[x_0, x_1, x_2]$ and a model ***f***

→ z' = [001, 011, 101, 010, 110, 100]

g (z') = ϕ_o + ϕ(001) + ϕ(011) ϕ(101) + ϕ(010) + ϕ(110) + ϕ(100) ~ f (x)

Shapley Values

- Lloyd Shapley (1923-2016)
  - Mathematician and Economist
  - Many contributions on Game Theory
  - Shapley Value introduced in his post-doctoral thesis.
- Solution concept in cooperative game theory
- Game Theory is the study of mathematical models of interaction among rational decision-makers.
- Defined by the author as “a mathematical study of conflict and cooperation”
- Numerical representation of the collaboration of players in a game to obtain some payout.

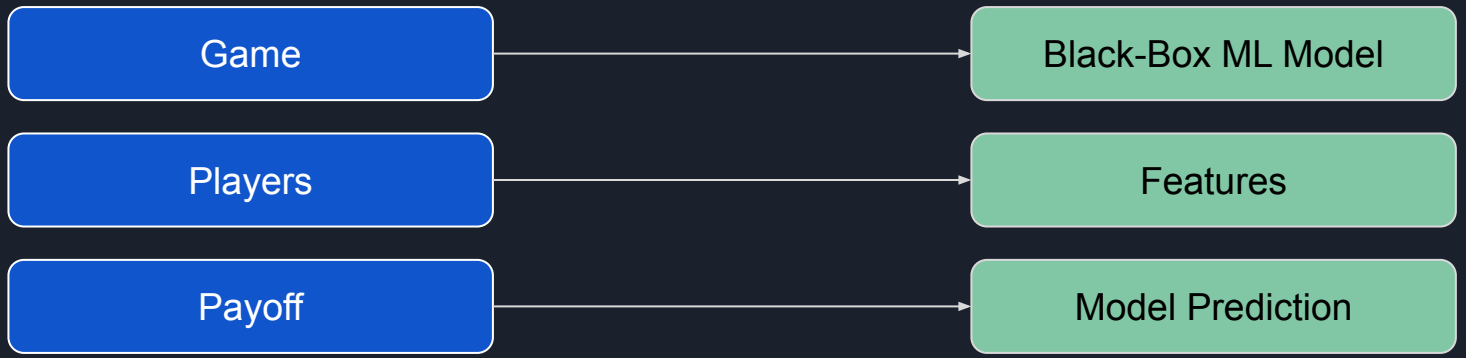
Shapley Values: Nomenclature

- N: number of players
- v: function that maps subsets of players (simplified features) to real numbers (gains) $v : 2^N \rightarrow R$, where $v(\varnothing) = 0$.
- S: coalition of players for which v(S) is the worth of the coalition of players and v(x), $x = \{p_1, ..., p_n\}$ where $p_i \in S$ describes the expected payoff for the group of players x.
- $\varphi(v)$ denotes the shapley value for the the gain function v

Shapley Values: Properties

- **Efficiency**: The sum of the Shapley values of all players equals the value of the grand coalition, so that all the gain is distributed among the agents.
- **Symmetry**: If i and j are two players who are equivalent in the sense that the addition of each player to a set contributes to the same gain $v(S \cup \{i\}) = v(S \cup \{j\})$, then $\varphi_i(v) = \varphi_j(v)$, their Shapley values are the same.
- **Linearity:** $\varphi_i(v+w) = \varphi_i(v) + \varphi_i(w)$ and $\varphi_i(av) = a^*\varphi_i(v)$
- **Null player:** If the gain is the same for a player group independent if another player is in the group, the Shapley value for that player is 0. If $v(S \cup \{i\}) = v(S)$ then $\varphi_i(v) = 0$
- **Anonymity**: The labeling of the players doesn't affect the assignation of their gain
- **Marginalism**: The Shapley value can be defined as a function which uses only the marginal contributions of player i as the arguments.

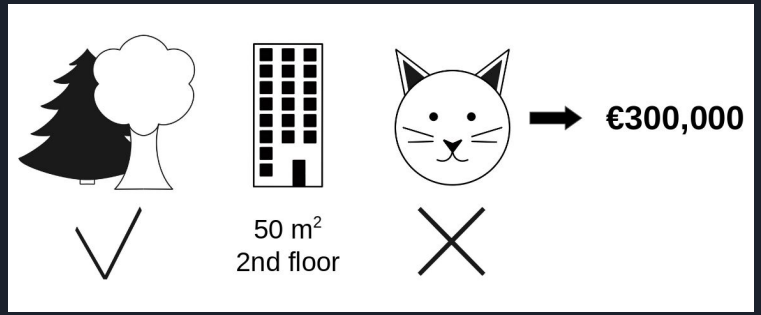
Shapley Values (in ML)



- A group of players cooperates and obtains some payoff in a game.
- A coalition of features cooperate to get a prediction from a ML model.

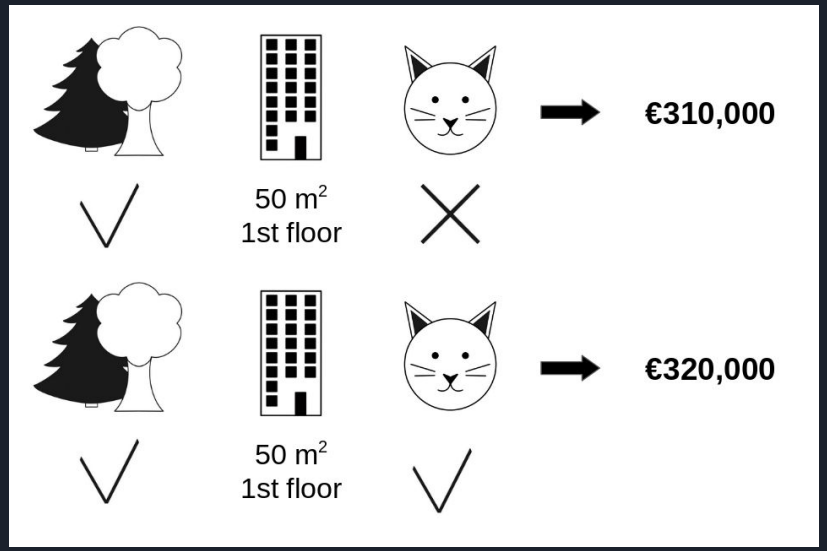
Shapley Values: Example

- Average for all apartments = 310k€
- Explain the difference of -10k€



Shapley Values: Example

- We simulate that only park-nearby, cat-banned and area-50 are in a coalition
- We try with cat-allowed as it was in the instance and replacing it with a random sample from the dataset.
- The contribution of cat-banned was €310,000 - €320,000 = -€10,000.



This operation is performed for all possible coalitions.

Shapley Values: Estimation

- All the sets of features have to be evaluated with and without the feature
- Exponential wrt. number of features
- Monte-Carlo Simulation

$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(x^m_{+j}) - \hat{f}(x^m_{-j}) \right)$$

$f(x^m_{+j})$ is the prediction for x with a random number of feature values replaced by values from the dataset for all features but *j*. The x^m_{-j} is almost identical but the *j* feature is also replaced.

Each of these *M* new instances is assembled from two instances.

Shapley Values: Estimation Algorithm

Algorithm 1:

Result: Shapley value for the value of the j-th feature
initialization;
Required: Number of iterations M, instance of interest x, feature index j, data matrix X, and machine learning model f

for *m=1,...,M:* **do**

Draw random instance z from the data matrix X;
Choose a random permutation o of the feature values;
Order instance x: $x_o = (x_{(1)}, \ldots, x_{(j)}, \ldots, x_{(p)})$;
Order instance z: $z_o = (z_{(1)}, \ldots, z_{(j)}, \ldots, z_{(p)})$;
Construct two new instances:

- With feature j: $x_{+j} = (x_{(1)}, \ldots, x_{(j-1)}, x_{(j)}, z_{(j+1)}, \ldots, z_{(p)})$
- Without feature j: $x_{-j} = (x_{(1)}, \ldots, x_{(j-1)}, z_{(j)}, z_{(j+1)}, \ldots, z_{(p)})$

Compute marginal contribution: $\phi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j})$;

end

Compute Shapley value as the average: $\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m$

Shapley Values: Advantages

fairly distributed

If two players are symmetric in two games, then their payoffs change by the same amount. Delivers a full explanation. LIME does not guarantee that.

contrastive explanations

A prediction can be compared to a single prediction as well as the entire dataset. Local models can’t do that.

solid theory

Not based on linear assumptions. Based on a theoretical background.

Shapley Values: Disadvantages

computing time	Not feasible in most of the real-world problems. Solution: limit the number of iterations.
misinterpretation	not the value lost by the absence of the feature. The difference between the prediction and the mean prediction.
no prediction model	It cannot be used to make statements about changes in prediction for changes in the input
inclusion of unrealistic data	when features are correlated. As when a feature value is missing from a coalition, we marginalize the feature.

SHAP

SHAP: Definition

- Goal: explain the prediction of an instance x_i by computing the contribution of each feature to the prediction.
- Shapley values tell us how to fairly distribute the gain among the features.
- A player can be an either an individual feature value or a group of feature values.
- Shapley values are represented as an additive feature attribution method.

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

where g is the explanation model, $z' \in \{0, 1\}^M$ is the simplified feature vector, M is feature combinations and φ_j is the Shapley value for the feature combination j.

SHAP: Coalition Vector / Simplified Feats

- an entry of 1 means that the corresponding feature value is "present" and 0 that it is "absent".
- To compute Shapley values, simulate that only some features values are playing ("present") and some are not ("absent").
- The representation as a linear model of coalitions is a trick for the computation of the ϕ's.
- For x, the instance of interest, the coalition vector x' is a vector of all 1's:

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

SHAP Python Package / SHAP Plots

- Python package Github repository
- SHAP Python package docs
- SHAP Plots Colab Notebook

Model Agnostic: Kernel SHAP

Kernel SHAP

- Work with a model regardless of the architecture or the type
- 5 steps:
  1. Sample coalitions
  2. convert z'_k to the original feature space and then applying model f.
  3. Compute the weight for each z'_k with the SHAP kernel.
  4. Fit weighted linear model.
  5. Return Shapley values φ_k, the coefficients from the linear model.

Kernel SHAP-1: Sample Coalitions

- binary vector of dimension P. 2^P possible coalitions for an exhaustive search.
- K coalitions are sampled and become the new dataset of simplified features.
- A regression model is trained.
- The target for the regression model is the prediction for a coalition.

Kernel SHAP-2: Convertion

- The model itself has not been trained in with the simplified features.
- They need to be translated to the dataset's domain.
- We define:

$$h_x(z'), h_x : \{0,1\}^P \rightarrow \mathbb{R}^P$$

- Function h_x maps:
  - 1 -> corresponding value from the instance x to be explained
  - 0 -> random instance in the dataset
- Why random?
  - ignoring dependency structure between present and absent features

Kernel SHAP-3: Weighting

- After applying the model f to the instance in the original space.
- Compute the weight for each z'_k with the SHAP kernel.
- Different from approaches as LIME.
  - LIME weights instances by proximity to the original sample (with no values replaced)
  - SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation.
  - Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights. This way the effects of each feature can be estimated isolatedly.

Kernel SHAP-3: Weighting

- Lundberg et. al propose the following SHAP kernel (weighting)

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

where M is the max number of coalitions and |z'| is the number of the present features on the instance z.

```
 M = 8
|z'| = 1        πx(z') = 0.125
|z'| = 2        πx(z') = 0.021
|z'| = 3        πx(z') = 0.008
|z'| = 4        πx(z') = 0.006
|z'| = 5        πx(z') = 0.008
|z'| = 6        πx(z') = 0.021
|z'| = 7        πx(z') = 0.125
```

Kernel SHAP-4: Train Regression Model

- We have the model, the target and the weights. We can now train the linear regression model.

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

- The ϕ values of the regression model will be the shapley values.
- The model is trained using squared errors.

SHAP Python Package / Kernel SHAP

- Kernel SHAP in Github Repository
- Brute force Kernel SHAP python implementation
- Kernel SHAP Colab Notebook

Model Dependant

Model Dependant

Differences:

- 100% accurate only for the particular model: Linear, Trees, Deep, etc.;

- Way faster than Kernel SHAP.

Linear SHAP

- Simplest SHAP approximation
- 100% accurate only for linear models

- Single feature representation of a Linear Regressor:

$$f(x) = \sum_{j=1}^{M} w_j x_j + b$$

- independent component of SHAP

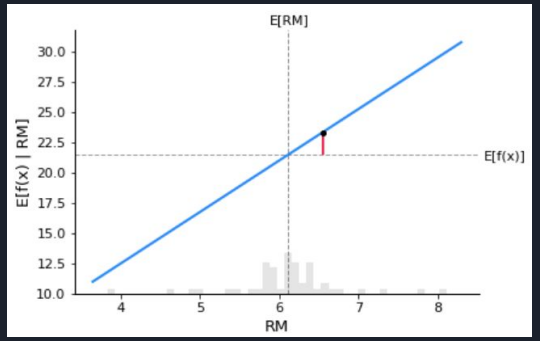
$$\phi_0(f, x) = b$$

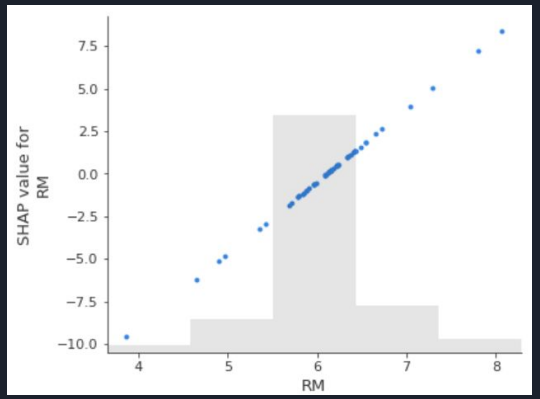
- The value for each shapley value is computed as a linear prediction where the Expected value of the feature matches ϕ=0

$$\phi_i(f, x) = w_j(x_j - E[x_j])$$

SHAP Python Package / Linear SHAP

- Linear SHAP in Github Repository
- Linear SHAP docs
- Linear SHAP Colab Notebook





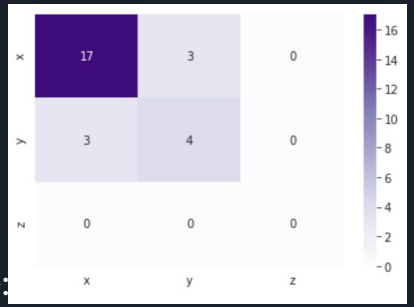
Tree SHAP

Introduction

- Compute exact SHAP values for tree-based machine learning models (decision trees, random forests and gradient boosted trees);
- It reduces the complexity to compute exact SHAP values from exponential to polynomial;
- Introduces a new approach that substitutes the listing decision paths to capture the feature interaction.

Tree SHAP

SHAP Interaction values



- Allocation of credit between all pairs of players;
- Instead of a vector, it provides a matrix of feature attributions where:
  - The main effects are on the diagonal
  - The interaction effects on the off-diagonal

  ⇒ SHAP value = diagonal + ∑ off-diagonal *

- Definition:

$$\Phi_{i,j} = \sum_{S \subseteq N \setminus \{i,j\}} \frac{|S|!(M - |S| - 2)!}{2(M-1)!} \nabla_{ij}(S),$$

- $\Phi_{i,j}(f, x) = \Phi_{j,i}(f, x)$
- * Total interaction effect is $\Phi_{i,j}(f, x) + \Phi_{j,i}(f, x)$

When i≠j, and

$$\begin{aligned}\nabla_{ij}(S) &= f_x(S \cup \{i,j\}) - f_x(S \cup \{i\}) - f_x(S \cup \{j\}) + f_x(S) \\ &= f_x(S \cup \{i,j\}) - f_x(S \cup \{j\}) - [f_x(S \cup \{i\}) - f_x(S)].\end{aligned}$$

Tree SHAP

Exponential Algorithm

Estimates $E[f(x) \mid x_S]$ recursively in $O(TL2^M)$, where:

- T is the number of trees;
- L is the maximum number of leaves in any tree;
- M is the number of features.

```
1:  procedure EXPVALUE(x, S, tree = {v, a, b, t, r, d})
2:      procedure G(j)                          ▷ Define the G procedure which we will call on line 10
3:          if v_j ≠ internal then              ▷ Check if node j is a leaf
4:              return v_j                      ▷ Return the leaf's value
5:          else
6:              if d_j ∈ S then                 ▷ Check if we are conditioning on this feature
7:                  return G(a_j) if x_{d_j} ≤ t_j else G(b_j)   ▷ Use the child on the decision path
8:              else
9:                  return [G(a_j)·r_{a_j} + G(b_j)·r_{b_j}]/r_j   ▷ Weight children by their coverage
10:     return G(1)                             ▷ Start at the root node
```

Tree SHAP

Exponential Algorithm → Polynomial

Estimates E [f(x) | x_S] recursively in $O(TL2^M)$, where:

- T is the number of trees;
- L is the maximum number of leaves in any tree;
- M is the number of features;
- D is the max depth of the trees.

$O(TLD^2)$

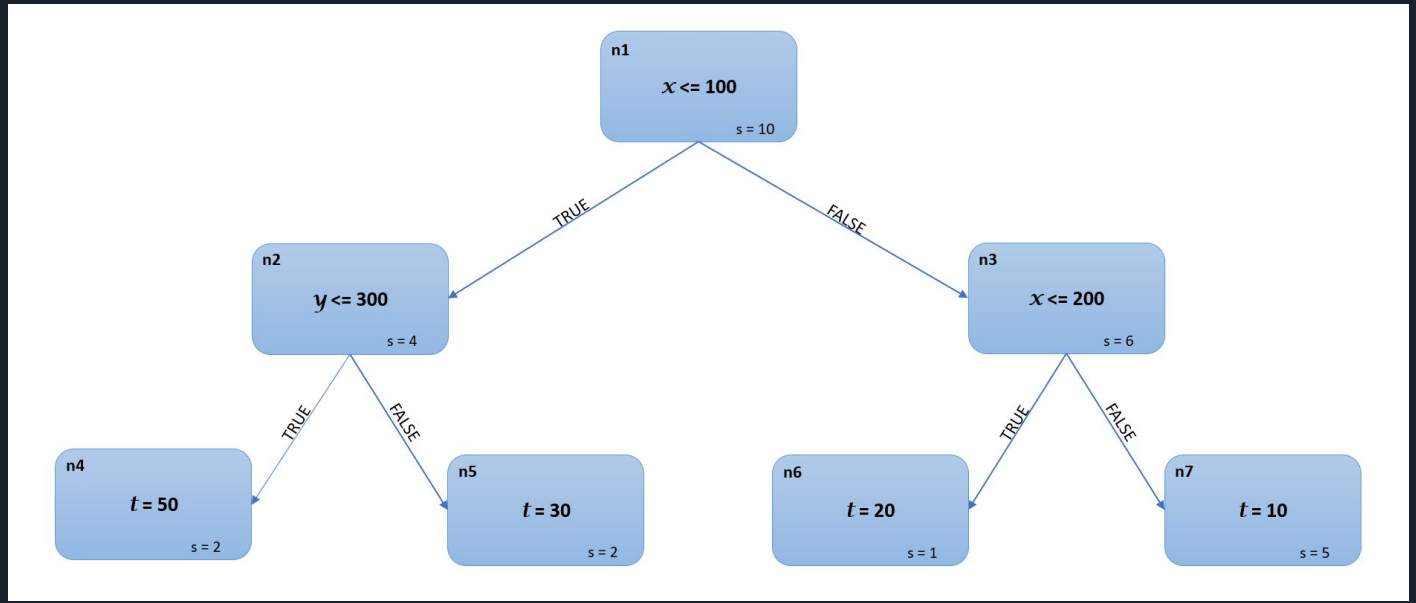
Like running the previous one but all 2^M subsets of S together

```
1:  procedure EXPVALUE(x, S, tree = {v, a, b, t, r, d})
2:      procedure G(j)                          ▷ Define the G procedure which we will call on line 10
3:          if v_j ≠ internal then              ▷ Check if node j is a leaf
4:              return v_j                      ▷ Return the leaf's value
5:          else
6:              if d_j ∈ S then                 ▷ Check if we are conditioning on this feature
7:                  return G(a_j) if x_{d_j} ≤ t_j else G(b_j)   ▷ Use the child on the decision path
8:              else
9:                  return [G(a_j)·r_{a_j} + G(b_j)·r_{b_j}]/r_j   ▷ Weight children by their coverage
10:     return G(1)                             ▷ Start at the root node
```

Tree SHAP

Example

[Tree SHAP-simple example Colab Notebook]



SHAP Python Package / Tree SHAP

- Tree SHAP in Github Repository
- Tree SHAP docs
- Tree SHAP Colab Notebook

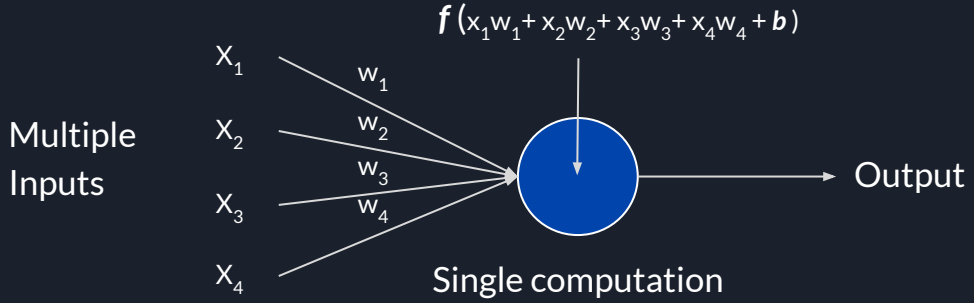
Deep SHAP

Introduction

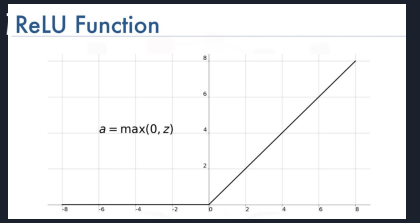
- Compute exact SHAP values for deep learning models;
- Deep SHAP = Deep Lift + Shapley values

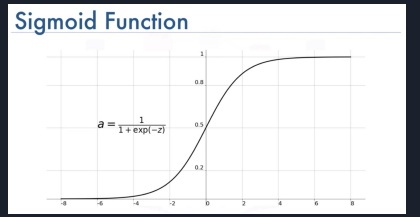
Deep learning intro

Artificial Neuron



Commonly used

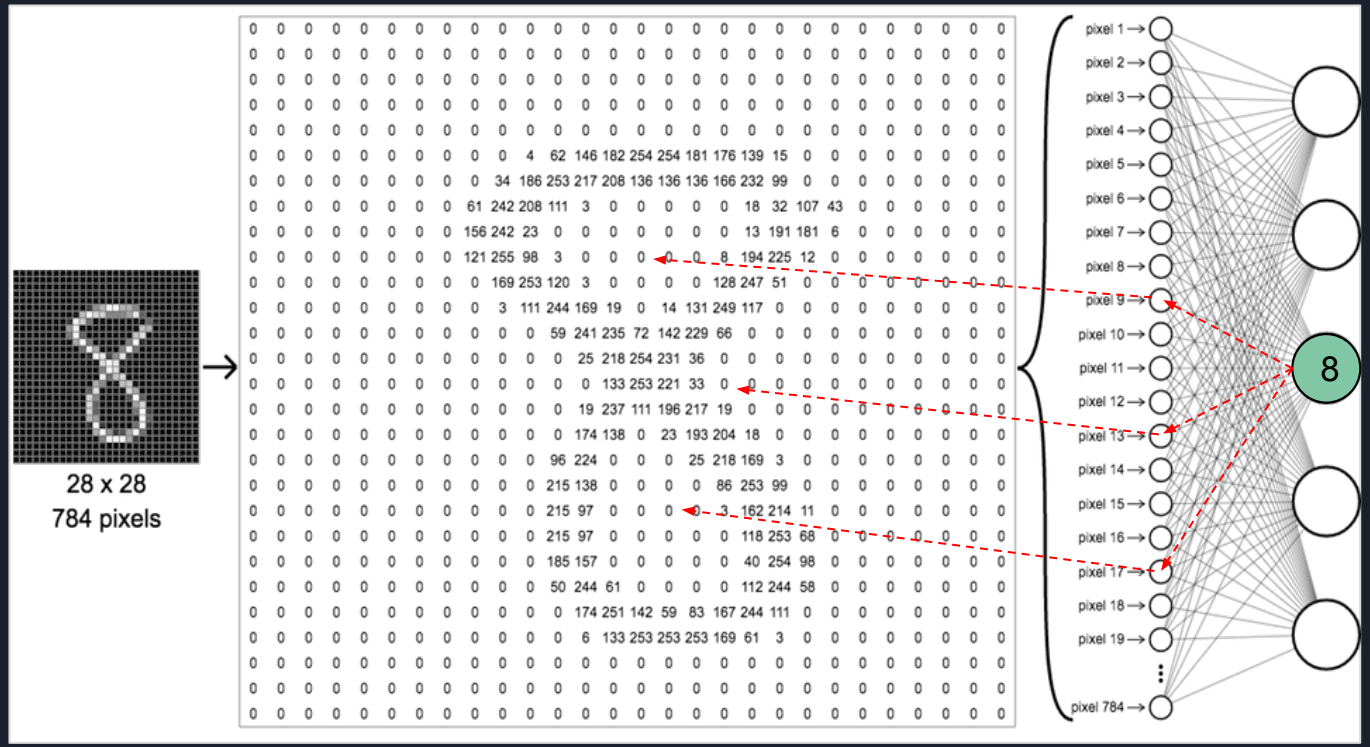




Deep SHAP

Explainable AI for Deep Learning

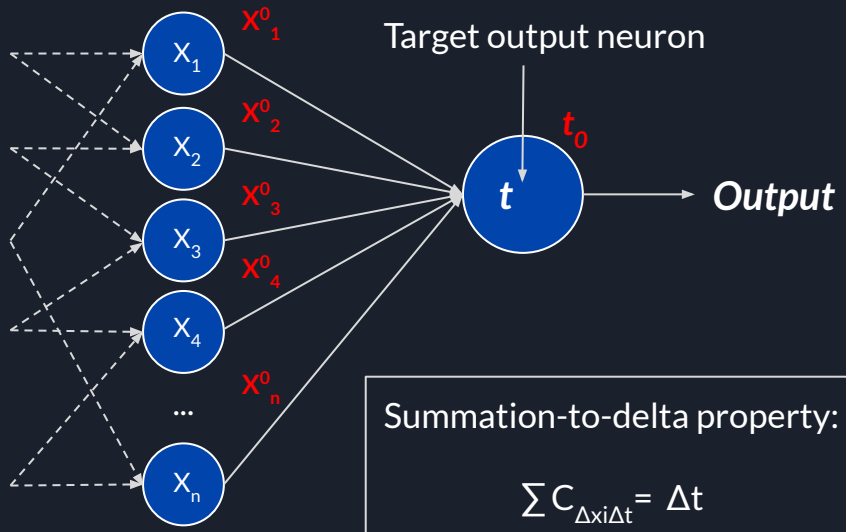
Goal: Example-specific explanations



DeepLIFT

Deep **L**earning **I**mportant **F**ea**T**ures

"Difference in the output from some 'reference' output, in term of difference of the input from some 'reference' of the input"



- t_0 reference activation of t.
- $x^0_1, x^0_2, \ldots, x^0_n$ reference neurons.

⇒ difference-from-reference of t

$\Delta t = t - t_0$

⇒ contribution scores of Δx_i

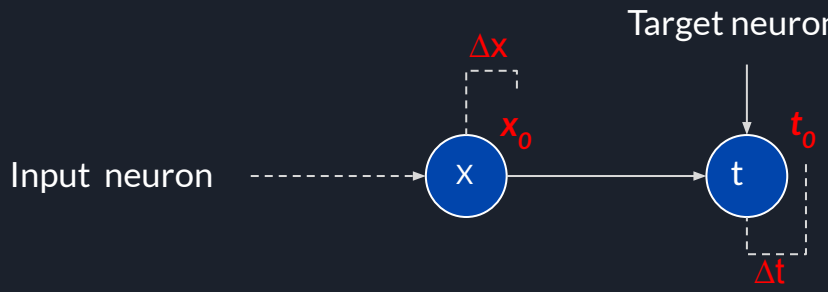
$C_{\Delta x_i \Delta t}$

Summation-to-delta property:

$\sum C_{\Delta x_i \Delta t} = \Delta t$

DeepLIFT

Deep **L**earning **I**mportant **F**ea**T**ures - Multipliers and the Chain Rule



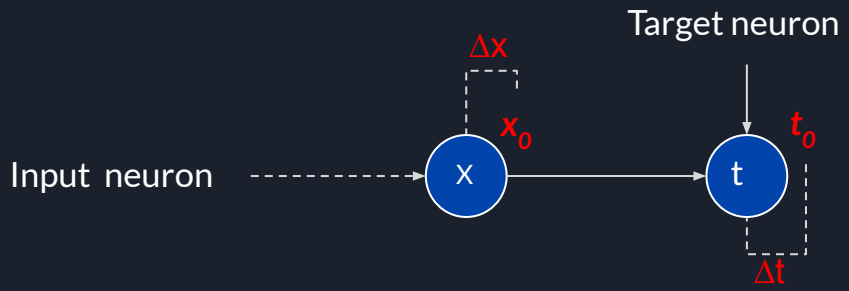
Multipliers

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \sim \partial t / \partial x$$

partial derivative (gradient)

DeepLIFT

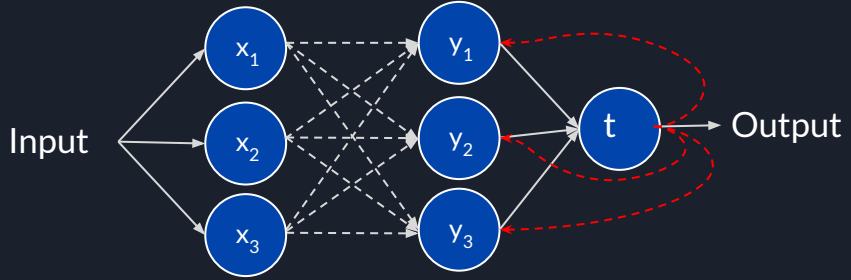
Deep **L**earning **I**mportant **F**ea**T**ures - Multipliers and the Chain Rule



Multipliers

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \sim \partial t / \partial x$$

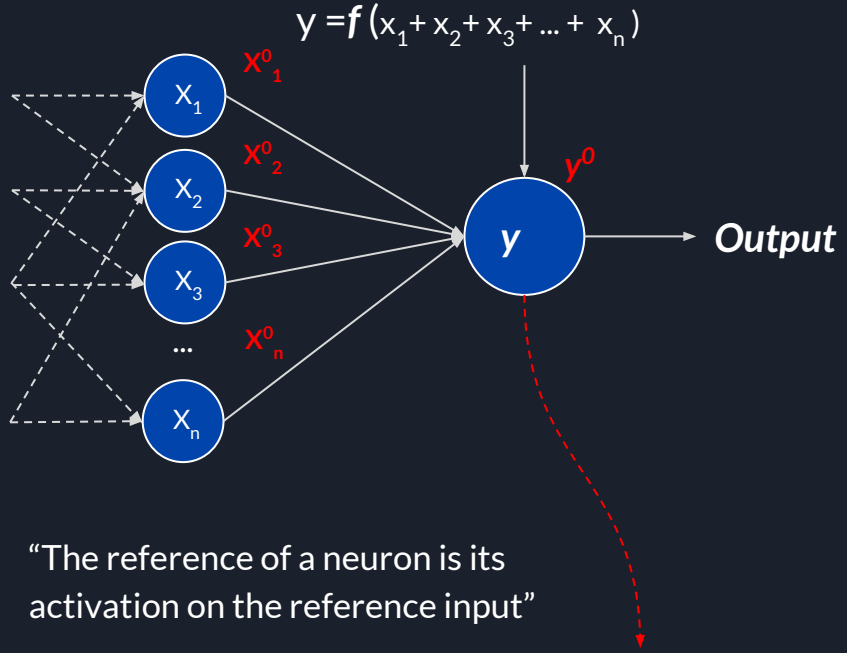
partial derivative (gradient)



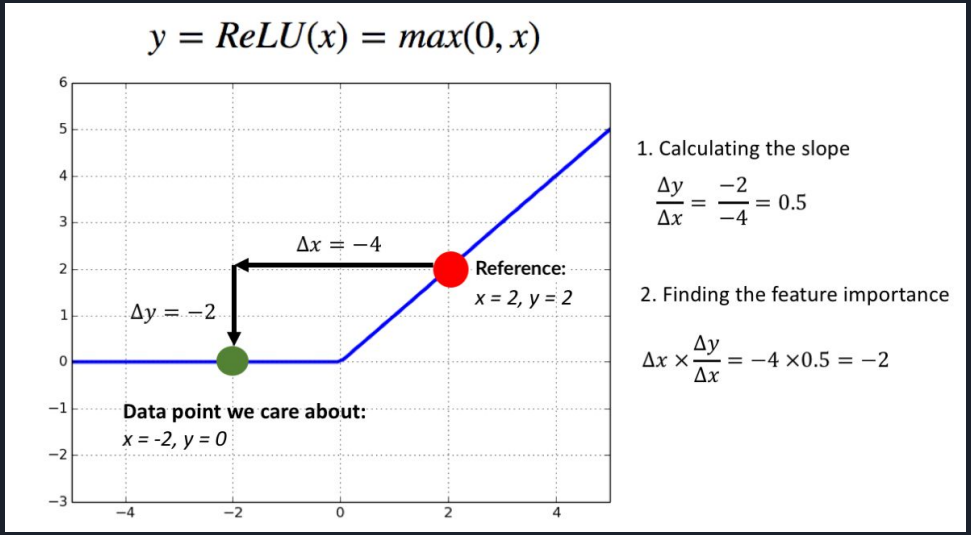
Chain Rule of Multipliers leads to explain backpropagation

DeepLIFT

Deep **L**earning **I**mportant **F**ea**T**ures - Example



"The reference of a neuron is its activation on the reference input"



DeepLIFT

to Deep SHAP

DeepLIFT is a back-propagation based approach that satisfy:

- Local accuracy
- Missingness

⇒ Shaley values represent the only attribution method that satisfies Consistency.

⇒ **Deep SHAP** extends DeepLIFT to approximate SHAP values.

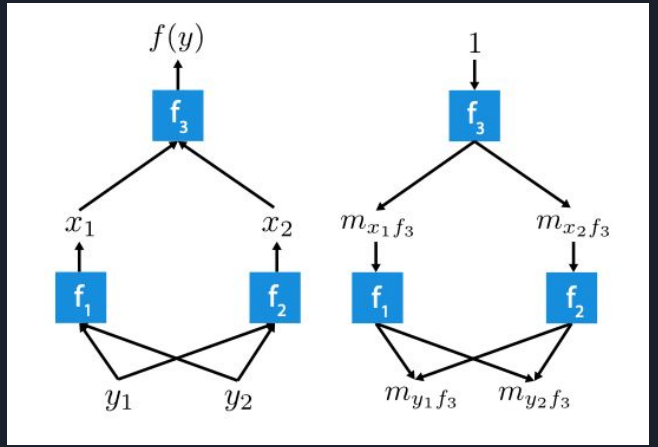
HOW?

Deep SHAP

Properties

- Deep neural networks are comprised of many simple components;
- Simple network components can be efficiently solved analytically if they are linear;
- For more complex components, It derives an effective linearization from the SHAP values computed for each component.

⇒ fast approximation of values for the whole model!



- DeepLIFT's multipliers, defined as SHAP values;
- Not using a single reference value but **a distribution of background samples.**

SHAP Python Package / Deep SHAP

- Deep SHAP in Github Repository
- Deep SHAP docs
- Deep SHAP Colab Notebook

References

1. Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: (2017).
2. Gabriel G. Erion Scott M. Lundberg and Su-In Lee. “Consistent Individualized Feature Attribution for Tree Ensembles”. In: (2018).
3. Avanti Shrikumar - Peyton Greenside - Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. In: (2019).
4. Andrea Masi & Victor Badenas. Google Drive notebook repositories. url: https://drive.google.com/drive/folders/1oNLP- BD9QhU7RIF0OIavjRBHBIdt2kmW?usp=sharing.
5. Scott M. Lundberg et al. “Explainable AI for Trees: From Local Explanations to Global Understanding”. In: (2019).
6. Scott Lundberg. An introduction to explainable AI with Shapley values. url: https://shap.readthedocs.io/en/latest/example_notebooks/overviews/An%20introduction%20to%20explainable%20AI%20with%20Shapley%20values.html.
7. Christoph Molnar. Interpretable ML book - Chapter 5.9: Shapley Values (2021). url: https://christophm.github.io/interpretable-ml-book/shapley.html.
8. Christoph Molnar. Interpretable ML book - Chapter 5.10: Shap (2021). url: https://christophm.github.io/interpretable-ml-book/shap.html
9. Christoph Molnar. Interpretable ML book - Chapter 5.10: Kernel Shap (2021). url: https://christophm.github.io/interpretable-ml-book/shap.html#kernelshap.

References

10. Christoph Molnar. Interpretable ML book - Chapter 5.10: Tree Shap (2021). url: https://christophm.github.io/interpretable-ml-book/shap.html#treeshap.
11. Janis Klaise & Alexandru Coca. Tree SHAP/Theoretical overview. url: https://github.com/SeldonIO/alibi/blob/a8efe1bdd8164bef28c3c22687caf4231ce587a5/doc/source/methods/TreeSHAP.ipynb.
12. Hugh Chen & Scott Lundberg & Su-In Lee. Understanding Shapley value explanation algorithms for trees. url: https://hughchen.github.io/its_blog/index.html.
13. Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning important features through propagating activation differences”. In: International Conference on Machine Learning. PMLR. 2017, pp. 3145–3153.
14. Scott Lundberg. Shap Python package. url: https://github.com/slundberg/shap.
15. Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In:Advances in Neural Information Processing Systems 30. Ed. by I. Guyonet al. Curran Associates, Inc., 2017, pp. 4765–4774. url: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.
16. Scott M. Lundberg et al. “From local explanations to global understanding with explainable AI for trees”. In: Nature Machine Intelligence 2.1 (2020), pp. 2522–5839.
17. Scott M Lundberg et al. “Explainable machine-learning predictions for the prevention of hypoxaemia during surgery”. In: Nature Biomedical Engineering 2.10 (2018), p. 749.


SHAP
SHapley Additive exPlanations
Thanks for your attention
May 2021