# Interpretable Nearest Neighbor Queries for Tree-Structured Data in Vector Databases of Graph-Neural Network Embeddings

Lukas Pfahler
lukas.pfahler@tu-dortmund.de
TU Dortmund University
Dortmund, Germany

Jan Richter
jan-philip.richter@tu-dortmund.de
TU Dortmund University
Dortmund, Germany

## ABSTRACT

We propose a method for interpreting similarity computations between neural embeddings of trees. We showcase our approach in a search engine for mathematical equations crawled from arxiv.org. The equations are encoded as MathML, a XML file format that describes math in tree structures. These trees are processed by a graph convolutional neural network to obtain fixed-size low dimensional and dense embedding vectors for each tree. Search is performed by performing nearest neighbor query in the set of embeddings. However just by the embeddings it is difficult to judge how the similarity came about. We propose two different approaches to highlight parts of the equation that are important for the similarity, one based on the forward-computation of the neural network and one based on the gradient in the backward-computation. In a qualitative study on math retrieval we show the advantages of both methods.

## KEYWORDS

neural networks, interpretability, information retrieval

## 1 INTRODUCTION

The current advances in artificial intelligence, particularly in deep learning, have had a huge impact on information retrieval and has changed the way we process, index and retrieve data. Artificial neural networks transform multimedia content like text, images, audio or video and compute latent representations which we can use to detect semantic similarities.

In this work we focus on retrieval of mathematical expressions. We showcase a neural-network based math retrieval engine that finds semantically related formulas based on contextual similarity. The system is designed to help scientists find related research based on math-queries. We leverage machine learning to process a large corpus of mathematical expressions and learn a vectorial representation that uncovers relations between equations. A human reader can use background knowledge on conventions and notations to infer the context of an mathematical expression or judge the relevance of an equation for a search query. We hope to learn a model that can perform this task using large amounts of data. Our system starts by representing formulas or equations in an XML-tree format, namely MathML, and uses a graph convolutional neural network to embed these trees in a dense, low-dimensional vector space that we can search efficiently using index data-structures for fast nearest neighbor retrieval. At the end of this machine learning pipeline, the users are interacting with the search interface and are presented with results. These results are based on neural network computations and it is often unclear how they came about. Traditional keyword queries are easily interpreted: We can highlight where the query-keywords or expressions related to the keywords appear in the results, thereby justifying the systems output. Because our system does not rely on exact matching of sub-terms, but rates overall semantic relatedness, the outputs are harder to interpret and harder to visualize, as we cannot necessarily highlight overlapping parts of result and query. To overcome this issue, we propose to apply so-called salience map for visualizing the similarity scores predicted by a graph convolutional neural network. These visualizations help both the end user in understanding how results came about and the machine learning engineer who deploys the models in understanding how the model rates similarities.

The rest of this work is structured as follows: We begin by discussing related work, including aspects of interpretable machine learning as well as geometric deep learning. Then we present our two methods for visualizing embeddings of trees in Section 3. In Section 4 we discuss our results in the context of a search engine for mathematical content. We discuss our dataset and data preparation and perform a qualitative study of our visualizations. We conclude this paper with an outlook in Section 5.

## 2 RELATED WORK

The call for interpretable machine learning models is popular in times where more and more decisions are automated using tools of artificial intelligence. It is connected to questions of accountability as well as ethics and morality and is often accompanied by dystopian visions of out-of-control artificial intelligence systems [9]. In practice, interpretable machine learning can roughly be grouped into two groups: interpretable model families and "model-agnostic interpretation tools" [9]. The former look at models that are interpretable [12]: Usually we mention decision trees and linear models here. However some restrictions apply: In order for either of them to be interpretable, the number of parameters they use must be sufficiently small and the interactions between the parameters must be easily understandable: A deep decision tree is no longer easy to interpret, even when it uses only a small number of features. A linear classification can be interpretable with a larger number of used features, as the features can be interpreted independently. A polynomial classification is more complicated to interpret in that regard. When decision trees become too complex to be interpreted, simpler models like decision lists or ordered decision lists offer a simpler solution [2] as they do not recursively branch. Many of the recent successes of machine learning models for practical problems come from the use of massively over-parametrized neural models. These are diametrically opposed to the two characteristics of interpretable models mentioned above, as they use millions of parameters and their non-linear nature obfuscates how these parameters interact.

Here we can apply the latter group of approaches, that either aim to make the blackbox model interpretable[13] or aim to make the *decisions* of black-box models interpretable (e.g. [5]). In the context of image classification the most popular techniques are based on visualizations where we highlight regions of the input image that are important for the overall classification. One of the popular approaches in image classification computes a derivative in the first layer and projects it onto the image [14]. Building onto the deconvolution technique[14, 18], Springenberg et al. propose a kind of guided backpropagation where negative gradients are set to zero in the backward step to focus solely on input features which increase the activation of a higher layer unit [15]. Another approach by Stylianou and Pless visualizes the output of a convolutional neural network using just the activations of the last convolutional layer to identify the regions of interest for the image classification [16]. We apply both directions for visualizations in a similarity learning task rather than a classification task and modify them to work with graph convolutional networks instead of plain convolutional neural networks.

Geometric Deep Learning is an umbrella term for deep neural model approaches for structured data like manifolds or graphs/trees. The term graph convolutional neural network is also prominent. The results allow the application of deep architectures to inputs that represent e.g. molecules [3], point-clouds [17] or social networks [6]. We work with XML-representations of math.

Finally we want to discuss our application example of information retrieval for mathematical expressions. Approaches in this area can be divided into two partitions: Approaches that use a sequential representation of the mathematical content and approaches that apply a tree-representation [7]. Similarity computation in tree-based approaches often requires computing dynamic programming algorithms that match subtrees, hence complex index data-structures are required for efficient retrieval. More recently Mansouri et al. proposed to embed mathematical expression in a dense vector space[7] based on a sequential representation of the equations. Pfahler et al. propose to embed equations using an image representation and regular convolutional neural networks that are trained to predict contextual similarity [11].

# 3 METHOD

We begin to describe our method by reviewing the definition of graph convolutional neural networks. Then we propose two different methods for obtaining visualizations of nearest neighbor queries: one based on forward information and one based on backward information.

## 3.1 Graph Neural Networks for Trees

We define tree structures $x = (X, E)$ as a tuple of node-features $X$ and edges $E$. Let $|x|$ denote the number of nodes in $x$. We assume that $X \in \mathbb{R}^{|x| \times d}$. A graph neural network maps a given tree to an output tree with transformed feature vectors in a $d'$-dimensional output space but with identical edge structure. We denote the neural network by $\phi(x) = (\phi(X, E), E)$. Let $\phi(x)_i \in \mathbb{R}^{d'}$ denote the output of the $i$-th node.

Graph neural networks are defined by composing different layers. Borrowing the notation of Morris et al. [10], an abstract graph network layer can be described by its output

$$x_i' = \psi\left(x_i, \square_{j \in \mathcal{N}(i)}\, \phi\left(x_i, x_j, e_{ij}\right)\right)$$

where $\phi, \psi$ are (sub-)differentiable operators such as linear transformations or multi-layer perceptrons, $\square$ denotes a differentiable, permutation invariant function like sum, mean or max and $\mathcal{N}(i)$ denotes the set of all neighboring nodes of $i$ in the tree with edges $E$. Note that $\phi$ might use information about the edges in the form of vectorial edge-features $e_{ij}$. An example of a simple graph convolutional layer is

$$x_i' = \sigma\left(W \sum_{j \in \mathcal{N}(i) \cup i} x_j\right)$$

which linearly transforms all nodes using a weight matrix $W$, aggregates by computing the sum of all neighborhoods and applying a component-wise activation function $\sigma$ like the sigmoid function or ReLU.

As long as all layers in a graph neural network are (sub-)differentiable operations, we can train the network via backpropagation. Efficient software libraries for training models with GPU-support are available, e.g. we use torch-geometric [4].

To store the fixed-size embedding of the whole tree in a vector database, we compute the mean of all nodes denoted by $\bar{\phi}(x) = |x|^{-1} \sum_i \phi(x)_i$. Then we can compute the cosine similarity or the dot product

$$\mathrm{sim}(x, x') := \langle \bar{\phi}(x), \bar{\phi}(x') \rangle$$

in order to obtain the similarity of two trees.

## 3.2 Forward-Visualization

As the embedding of a tree is the mean of all its nodes, we can see that each pair of nodes contributes to the overall similarity. The similarity i.e. dot-product can be decomposed as

$$\langle \bar{\phi}(x), \bar{\phi}(x') \rangle = (|x| \cdot |x'|)^{-1} \sum_{i=1}^{|x|} \sum_{j=1}^{|x'|} \langle \phi(x)_i, \phi(x')_j \rangle.$$

Following the ideas of Stylianou and Pless [16], we visualize the similarity by coloring the nodes according to their contributions to the overall similarity. Hence given the embedding of a query $\vec{q}$, we color the $i$-th node proportional to

$$c_i \sim \langle \phi_i(x), \vec{q} \rangle.$$

This approach is computationally cheap, we just have to run one forward pass for every search result we want to visualize.

## 3.3 Backward-Visualization

Following the ideas of salience maps, we color nodes in the tree according to the gradient of the similarity. Looking at a first order Taylor approximation of the similarity, we can write

$$\mathrm{sim}(\vec{x} + \Delta, \vec{q}) \approx \mathrm{sim}(\vec{x}, \vec{q}) + \langle \Delta, \nabla_{\vec{x}} \mathrm{sim}(\vec{x}, \vec{q}) \rangle$$

to approximate how the similarity changes based on perturbations $\Delta$ to the input $\vec{x}$. If we decrease the value of $x$ in the $i$-th coordinate, the corresponding coordinate in the gradient $\nabla_{\vec{x}}$ indicates whether the first-order approximation of the similarity decreases or increases. If the gradient is positive, then reducing the feature reduces the similarity. Thus the feature is important for the overall similarity. Hence we use the values in the gradient $\nabla_{\vec{x}}$ to visualize similarity.

In classical convolutional neural networks for image processing, the input is either a greyscale or an RGB-image. Thus at every position, there is either a 1-dimensional or 3-dimensional feature vector likewise gradient vector. If there is only one dimension, we do not need to aggregate the gradient, in the case

of three dimensions we often aggregate by using the maximum gradient over all color channels.

In our case, we have higher-dimensional feature vectors in $\mathbb{R}^d$. Hence we have to aggregate the gradient

$$c_i \sim f(\nabla_{x_i} \mathrm{sim}(x, q)).$$

where $f$ is an aggregation such as max, mean or the sum of all non-negative gradients.

We are particularly interested in the case where $x$ is a one-hot encoding of XML-data. Hence $x$ is sparse and binary. We propose to use only gradient information of components that are set to 1 in the input to answer the question how the similarity changes if we flip inputs from one to zero. The component-wise multiplication of the input and the gradient eliminates all gradient mass at input components that are zero. We aggregate only the remaining components

$$c_i \sim f(x_i \odot \nabla_{x_i} \mathrm{sim}(\vec{x}, \vec{q}))$$

and color the nodes proportional to this aggregation.

# 4 INTERPRETABLE SEARCH FOR FORMULAS

We showcase our approach in a semantic search engine for mathematical equations which we have crawled from arxiv.org. First we give details on our dataset and machine learning model. Then we perform a number of example queries and discuss the quality of our visualizations.

## 4.1 Data

We outline how we gather data from arxiv.org, transform them to tree structured data and encode them into vectorial embeddings.

*4.1.1 Dataset.* We are working on data obtained from arxiv.org, a service where scientists can upload their manuscripts or preprints without reviewing process. We have downloaded all publications up to April 2019 and filtered all publications that use the subject code `cs.LG` which covers computer science publications on machine learning. The publications are available as the original LaTeX-files.

Of these 9,936 publications, we sample two subsets, train and test of size 7,949 and 1,987 respectively. We use the train-set for building our index and the test-set as search queries.

From all publications, we extract mathematical expressions by using regular expressions for the most common math-environments like 'equation', 'align', etc. Furthermore we extract user-defined commands and macros. Using the library Katex[1] we compile the raw LaTeX-equations to the XML-based MathML format[2], originally designed for displaying and storing mathematical content in the semantic web. Since the equations are now in XML-format, we have to work with tree-structured data.

An earlier version of the dataset used in [11] is published at https://whadup.github.io/EquationLearning/ and the version used in this study will be made available as well.

*4.1.2 Data-Representation.* We see an example of an equation encoded as MathML in Figure 1. The MathML standard defines 30 different XML-tags like `<mi>` for math identifiers or `<mo>` for math operators. Some of these tags use attributes, for instance to change font or spacing. Leaf nodes contain text like numbers, parenthesis or letters (greek, latin, etc...). We encode each node

[1] http://katex.org

[2] More specifically, we use the Presentation-MathML format for displaying maths on the web as specified at https://www.w3.org/TR/MathML3/

```
<math xmlns="http://www.w3.org/...">
  <semantics>
    <mrow>
      <mfrac>
        <mn>1</mn><mi>n</mi>
      </mfrac>
      <msubsup>
        <mo>Σ</mo>
        <mrow>
          <mi>i</mi><mo>=</mo><mn>1</mn>
        </mrow>
        <mi>n</mi>
      </msubsup>
      <mi mathvariant="normal">ℓ</mi>
      <mo stretchy="false">(</mo>
      <mi>f</mi>
      <mo stretchy="false">(</mo>
      <msub>
        <mi>x</mi><mi>i</mi>
      </msub>
      <mo stretchy="false">)</mo>
      <mo separator="true">,</mo>
      <msub>
          <mi>y</mi><mi>i</mi>
      </msub>
      <mo stretchy="false">)</mo>
      <mo separator="true">,</mo>
    </mrow>
  </semantics>
</math>
```

**Figure 1: Example MathML for $\frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$.**

of the XML into one-hot-encoded vectors. We distinguish three groups of features: *tag* for encoding the XML-tag (e.g `<mi>`), *attribute* for encoding optional attributes (e.g. `stretchy="false"`) and *content* for the text in leaf nodes (e.g. `x`,`y`, `i`, etc.). We encode this data using a total of 256 features where we use 32 dimensions to encode the tags, 32 dimensions to encode the most frequent attributes (including one special symbol for unknown attributes) and the remaining 192 dimensions to encode the most frequent characters of the content (also including one unknown symbol). In addition to the one-hot encoded features, each node has an attribute that specifies the position within the parent node. We assume that edges in the tree are bidirectional.

*4.1.3 Embedding Network.* We use a graph convolutional network to map the 256-dimensional tree to a 64-dimensional tree and compute a fixed-size embedding by averaging over all nodes. We use a network architecture with 5 layers that use a hidden dimensionality of 256. Each layer computes a linear transformation of the inputs, computes the mean over all neighborhoods ($\square$ = mean) and applies the ReLU activation to the outputs. After each layer, we use batch normalization.

The network is trained using the similarity task proposed by Pfahler et al. [11] that proposes to rate two equations as similar when they appear in the same paper. Using this cheap proxy label for similarity, we can train a Siamese network [1, 8]. The detailed procedures for training the embedding network are beyond the scope of this work.

## 4.2 Qualitative Study

In this section we present a qualitative study of our proposed method in the context of math retrieval.

*4.2.1 Rendering and Color Palettes.* First we want to discuss some peculiarities of MathML trees. The advantage of these trees is that the underlying XML-format of our representation is a mark-up language. Hence there is a native way to visualize them

$$f(x) = f(x_0) + (x - x_0)^T \nabla f(x_0)$$
$$g(x) = f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2}(x - x_0)^T H (x - x_0)$$
$$g(x) = f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2}(x - x_0)^T H (x - x_0)$$

**Example 1: Example Query (Taylor Approximation)**

$$\frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$
$$\widehat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} l(f, x_i, y_i)$$
$$\widehat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} l(f, x_i, y_i)$$

**Example 2: Example Query (Empirical Risk Minimization)**

$$h^{l+1} := \sigma^l(W h^l + b)$$
$$z^l = f(W^l z^{l-1} + b^l)$$
$$z^l = f(W^l z^{l-1} + b^l)$$

**Example 3: Example Query (Multi-Layer Perceptron)**

by rendering the equation with a corresponding markup processor like Katex or MathJax[3]. We can add additional style information to the original markup to color the nodes according to our computed visualization. This way we obtain beautiful visualizations without worrying about issues like tree layouting etc. One problem with this approach is that we can only visualize some of the nodes. Particularly, with some exceptions like root or fraction bars, we will color only leaf-nodes, as most inner nodes belong to XML-tags that do not directly output visible symbols. For instance in Figure 1, the <msubsup>-tag corresponds to a node that does not output any symbols on its own, but only encapsulates the symbols of its children.

One way to mitigate this issue is to set the color intensity of the $i$-th node to the sum of all colors intensities on the path from root to $i$-th node. However we have seen that this puts too much emphasis on nodes that are deeper in the tree. Hence we currently omit all hidden nodes from the visualization.

We use max-aggregation of the gradients and compute colors by linear interpolation where black font indicates the most significant parts and grey parts are less significant.

*4.2.2 Results.* We find semantically related equations by applying annoy to create an index structure for nearest neighbor retrieval with dot-product similarity. The retrieved equations will not be equivalent or strictly equal, but depicting similar concepts. We analyze the visualizations for the nearest neighbor as output by the index. We compare the visualization based on the forward pass with visualization based on backward. In Examples 1-6 we present pairs of query (first row) and nearest neighbor and color the result according to the forward pass (second row) and backward pass (third row). All examples are from the context of machine learning, which allows us to judge the results based on our background-knowledge of the research area. We now discuss each example in detail:

[3]http://mathjax.org

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$$
$$\Delta_f(r|S) \triangleq f(S \cup \{r\}) - f(S),$$
$$\Delta_f(r|S) \triangleq f(S \cup \{r\}) - f(S),$$

**Example 4: Example Query (Submodular Functions)**

$$\mathbb{E}_\sigma \sup_f \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i)$$
$$\mathscr{R}_n(\mathscr{F}) = \mathbb{E} \sup_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i f(x_i)$$
$$\mathscr{R}_n(\mathscr{F}) = \mathbb{E} \sup_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i f(x_i)$$

**Example 5: Example Query (Rademacher Complexity)**

$$P(\hat{X} - \mathbf{E}[X] \geq t) \leq e^{-2nt^2}$$
$$\Pr(\overline{X} - E[\overline{X}] \geq t) \leq e^{-2nt^2} \Pr(\overline{X} - E[\overline{X}] \leq t) \leq e^{-2nt^2}$$
$$\Pr(\overline{X} - E[\overline{X}] \geq t) \leq e^{-2nt^2} \Pr(\overline{X} - E[\overline{X}] \leq t) \leq e^{-2nt^2}$$

**Example 6: Example Query (Hoeffding's Bound)**

In Example 1, we have queried the definition of the first-order Taylor approximation. The embedding model retrieved the second-order Taylor approximation as nearest neighbor. Both visualization put the emphasis on the $\nabla$-symbol, the symbol commonly used for gradients. Interestingly, the backward visualization indicates that the $H$ symbol, commonly used for the Hessian matrix i.e. the second order derivative, also contributes to the overall similarity score. Apparently the model has learned that both symbols appear frequently together, e.g. in the context of differentiable functions.

The query in Example 2 is the empirical risk functional commonly used in machine learning. Both approaches highlight that we have a sum over objects subscripted by $i$. In addition, the backward approach highlights the $\widehat{\cdot}$-symbol commonly used to indicate that a random quantity is estimated using a finite sample. The model seems to have learned that it frequently co-occurs with estimators of the form $\frac{1}{n}\sum_{i=1}^{N} z_i$.

Example 3 shows a query for the definition of a fully connected neural network layer. The forward-visualization focusses on the variables $W, B$ for weight matrix and bias vector, while the backward variant focusses on the superscripted index indicating the number of the layer $l$.

In Example 4, we see the definition of gain used in the context of submodular function maximization, where both visualizations highlight the $\cup$ symbol. There is almost no difference between the two variants, most notable is the stronger emphasis on the opening curly-brace in the forward-visualization.

For Example 5, we have queried the definition of the empirical Rademacher complexity, a measure used e.g. in statistical learning theory. Both approaches highlight the sup operation, which is indeed essential for the definition. Without the sup, the expected value would trivially amount to 0. The forward visualization seems to highlight the combination of expectation and supremum, whereas the backward variant highlights mostly supremum and $\varepsilon_i$. This indicates that the model has correctly learned that both $\sigma$ and $\varepsilon$ are frequently used for Rademacher averages.

The last Example 6 shows concentration inequality, more precisely Hoeffding's bound that bound the deviation from a sample

mean to the true expected value of a random variable $X$. Both visualizations highlight the $\bar{\cdot}$ symbol that is frequently used to indicate sample means and hence is related to the $\hat{\cdot}$ symbol for empirically estimated quantities used by the query. Additionally both visualization highlight the $X$ that is generally used for random variables in statistics literature. It is interesting to note that the forward vizualization also places emphasis on the $\leq$ operator, while the backward visualization mostly highlights the $\geq$.

Overall we see that the forward visualizations spread the emphasis more evenly over the tokens than the backward pass. We think that with each layer of the encoder network the information in the nodes becomes less local and more global, as the feature vector at each node is a function of all its neighbors on the previous layer. This mixing makes it more difficult to highlight where the important information is originally coming from. The backward-based approach does not have this issue. Both approaches put a lot of emphasis on math operators like equal or plus-signs.

## 5 CONCLUSION AND OUTLOOK

In this work we have proposed two different approaches for visualizing similarity computations between embeddings computed by graph convolutional networks. We have applied our method in the domain of math retrieval. Both methods have allowed us to gain insights into the way our model scores similarities. The visualizations indicate that our model was able to learn conventions and notations to infer the context in which equations appear.

We are confident that the methods proposed in this work are also useful in other application domains. For instance when we work with graph representations of molecules [3] and use geometric deep learning to solve classification or metric learning tasks, we can color two-dimensional or three-dimensional illustrations of the molecules using our methods.

In the future we want to also visualize the alignment between query and result. A possibility is building interactive visualizations that allow us to focus on individual symbols of the equation and highlight how the model judges the similarity for each individual token.

## ACKNOWLEDGMENTS

This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis".

## REFERENCES

[1] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. 2016. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith (Eds.).

[2] Chaofan Chen and Cynthia Rudin. 2018. An Optimization Approach to Learning Falling Rule Lists. In *International Conference on Artificial Intelligence and Statistics*.

[3] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[4] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[5] Riccardo Guidotti, Anna Monreale, Stan Matwin, and Dino Pedreschi. [n.d.]. Black Box Explanation by Learning Image Exemplars in the Latent Feature Space. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019*.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[7] Behrooz Mansouri, Douglas W Oard, C Lee Giles, and Richard Zanibbi. [n.d.]. Tangent-CFT : An Embedding Model for Mathematical Formulas. ([n. d.]).

[8] Hossein Mobahi, Ronan Collobert, and Jason Weston. 2009. Deep Learning from Temporal Coherence in Video. *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)* (2009), 737–744.

[9] Christoph Molnar. 2019. *Interpretable Machine Learning*.

[10] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[11] Lukas Pfahler, Jonathan Schill, and Katharina Morik. 2019. The Search for Equations – Learning to Identify Similarities between Mathematical Expressions. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019*.

[12] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206–215.

[13] Stefan Rüping. 2006. Learning interpretable models. (2006).

[14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR* (Dec. 2013).

[15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. *Striving for simplicity: The all convolutional net*. Technical Report.

[16] Abby Stylianou and Robert Pless. 2019. *Visualizing Deep Similarity Networks*. Technical Report. arXiv:arXiv:1901.00536v1

[17] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)* 38, 5 (2019), 1–12.

[18] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, 818–833.