

# Using Knowledge Graphs to Measure and Characterize the Interpretability of Deep Learning Interpretations

Mathieu d’Aquin

Liudmyla Klochko

mathieu.daquin@loria.fr

liudmyla.klochko@loria.fr

LORIA, Université de Lorraine, CNRS

Nancy, France

## Abstract

Mechanistic interpretability is a family of methods that try to derive interpretable features from deep learning models. The idea is that they exploit patterns in the activation vectors or connections of a neural network to extract values that can be expected to represent particular concepts, notions, measures, or mechanisms on which the network relies to derive its predictions. However, a key issue with these methods is that establishing the supposed meaning of those features, and therefore evaluating their actual interpretability, is generally a manual and subjective task. Based on the idea that the process of interpretation can be seen as one relating emergent features of the model to the knowledge of the domain, in this article, we propose a method relying on knowledge graphs to build and assess possible definitions of those features. We achieve that by creating decision trees classifying input data as having or not having the candidate interpretable feature and that are built using the properties of a related knowledge graph. The accuracy of those decision trees therefore gives us a measure of the interpretability of the extracted features with respect to a given knowledge graph, and the decision tree itself provides an approximate definition of the feature using the knowledge graph’s vocabulary. We illustrate the approach by using two different interpretability methods with a simple example relying on DBpedia, and show results obtained on a more complex case study in material science, using a more specialized knowledge graph.

## Keywords

mechanistic interpretability, evaluation, knowledge-based interpretation, knowledge graphs

## 1 Introduction

The need for machine learning models, specifically deep learning models, to be interpretable stems from multiple reasons: it might improve their safety by providing verifiable results; improve confidence in those results; support validating and debugging models; or even help identify new, unknown phenomena in the data used to train those models. Although many techniques and approaches to interpretability exist, mechanistic interpretability (see, for example, [15]) has the advantage that it directly relies on the mechanisms (the inner workings) of the model, and therefore provides internal interpretations of the model itself rather than an external interpretation of its behavior. It generally relies on analyzing/processing the activation vectors or connections within a

neural network to extract patterns or measures that are expected to correspond to interpretable features. In machine vision, this might, for example, relate to identifying that a sub-part of the network is specifically dedicated to identifying edges, angles, or other visual features [9].

A wide range of such methods has been proposed in the literature, including those based on training models to recognize specific features in the activations of a machine vision model [11], on dictionary learning methods to obtain sparse, positively defined values that are supposed to map to interpretable notions [10, 17], or on mapping activation networks in specific layers in an attempt to test the presence of the representation of specific concepts within the model [6]. The most visible and discussed objective of applying such methods is that of safety, especially when applied in the context of large language models [2], but it has also been argued that they could be used to support model debugging [12] or scientific discovery [8].

However, a key issue with these methods is that, while they are expected to extract features that are more *interpretable* than the activations or connections of the base model, those still need to be manually *interpreted*. In other words, while a measure extracted, for example, using a dictionary learning method as in [17] might be expected to represent an interpretable feature, a manual inspection of this measure on samples of relevant data is still necessary to understand its meaning, how it relates to existing descriptors in the domain, or how it can be labeled, and therefore how truly interpretable it is.

In this paper, we relate the idea of interpretation to that of aligning the interpretable units obtained from mechanistic interpretability methods to existing knowledge in the domain of the interpreted model. More precisely, we show how mapping such units to properties or combinations of properties in a relevant knowledge graph can provide a way to characterize the unit in question, as well as an approximated domain-relevant definition. By measuring the accuracy of those mappings, we therefore also provide a measure of the true interpretability of those units in the context of the given knowledge graph.

In order to establish those mappings, we train shallow decision trees to distinguish between data samples that lead to high activations of the corresponding units. The datasets to train those decision trees are formed by the values of properties in the neighborhood of the relevant data samples in a given knowledge graph. Besides their inherent explainability, the advantage of shallow (i.e. low depth) decision trees is that they automatically select the most appropriate properties on which to rely to obtain high accuracy,

as well as the thresholds to employ for numerical properties. We illustrate the method with a small example of a text classification model based on the DBpedia knowledge graph. We also show experiments carried out with a larger example of a graph neural network (GNN) used for regression on the atomic structure of materials (to predict their formation energy) by using a knowledge graph describing the physical properties of thousands of materials.

In these experiments, we test two approaches for mechanistic interpretability and compare them to a baseline. The baseline consists of simply using the activation vectors of layers in the GNN model as interpretable features. The two tested methods are based on training self-organizing maps (SOMs) on the activation vectors of different layers of the model as in [6] and on training sparse autoencoders (SAEs) on the activation vectors of different layers of the model as in [10, 17]. We show how those methods obtain different levels of interpretability on different layers of the network and how the interpretable units they identify strongly align with specific properties of materials, as found in the knowledge graph.

## 2 Related Work on Mechanistic Interpretability

Although deep learning and the increasing reliance on deep neural networks have led to great performance and impressive results in many fields, their applicability has been hampered by the fact that those results are difficult to verify, validate, and justify. The objective of interpretable AI [20] is to alleviate this issue by deploying techniques that can help a user gain a greater understanding of the way a model obtains its results and the aspects of the data on which it relies to do so. Among the many approaches to interpretability, we can find “knowledge distillation” (reducing a model to a simpler, more interpretable one [13]) or the use of feature-based explainability methods globally [19], including by abstracting them through knowledge graphs [14, 18]. However, these methods are external to the model. They do not interpret the model itself but offer plausible interpretations of its behavior as externally observable through its inputs and outputs.

Mechanistic interpretability takes a fundamentally different approach as it aims to provide interpretable units extracted directly from the inner workings of a (generally neural) model [16]. It has recently increased in visibility due to its applicability (although with many engineering challenges) to large language models and the hope that they could help identify the source of their emergent capabilities [15]. Recent surveys [15, 16] have identified categories of approaches and techniques for achieving mechanistic interpretability, including, in particular, the abstraction of subnetworks within the neural network and the abstraction of features as represented by activations and weights within the network. Circuit finding is an example of the former, where modules are identified that appear to carry a particular function within the model (see, for example, [4]). Recent works carried out with LLMs on the topic of “monosemanticity” are an example of the latter, where methods such as dictionary learning are used to disentangle the way neurons play a role in the representation of multiple features (polysemanticity), to identify those features and analyze how they contribute to the results (see, for example, [10, 17]). In this article, we test two methods that are more specifically related to this

last category of approaches. Indeed, a key issue with such methods is that, while they identify supposedly interpretable features, those still need to be validated and related to some meaning understandable by users. This is generally achieved manually, and there is a need to measure the actual interpretability of those features by providing them with processable interpretations. This is what we are proposing to do here by aligning them with properties of knowledge graphs. In that sense, this work shares common objectives both with methods aiming to quantify the interpretability of neural networks’ latent spaces [1, 21] and those aiming to automatically label extracted interpretable features [5, 7].

## 3 Tested Methods for Mechanistic Interpretability

As mentioned above, in this article, we propose a method to align interpretable units extracted from mechanistic interpretability approaches that are intended to identify interpretable features within the activation vectors of neural networks. We first introduce a simple example to illustrate those methods and the approach proposed in this article.

### 3.1 Introduction to the illustrative example

The illustrative example used here relies on a new version of a model already used in [6] to classify texts corresponding to biographies of painters (obtained from DBpedia) based on whether or not those painters have paintings exhibited in at least one major museum.<sup>1</sup> The training of the model itself is not relevant here, since the process of interpretability is independent from it and from the performance of the model itself (although it seems reasonable to think that interpretability results will be different in a highly precise model, from those of a less precise model with many biases). The key information for the remainder of this article is the architecture of the network itself. It simply consists of a classification layer added on top of the pre-trained DistilBERT model.<sup>2</sup> The whole model is trained on the painter dataset from DBpedia, therefore fine-tuning DistilBERT for the task of text classification on painters’ biographies.

### 3.2 Baseline: Using the base activation vectors of the model

The baseline used here is not truly an interpretability method and is used as a reference for comparison. If a problem is sufficiently simple and the model is large compared to the complexity of the problem, it could be possible that individual units within the neural model are specialized to represent specific features. This baseline method, therefore, consists of considering the units within the neural network directly as representing candidate interpretable features, with the activation values of those units being correlated with the presence of those features.

Although this method is the simplest, there are some subtleties to consider in its realization. For example, in our painter model, some layers are simple linear layers. Therefore, in those cases, for each data point in the dataset, an activation vector can be obtained

<sup>1</sup>the code to train it can be found at <https://github.com/mdaquin/PainterModel/>

<sup>2</sup>[https://huggingface.co/docs/transformers/en/model\\_doc/distilbert](https://huggingface.co/docs/transformers/en/model_doc/distilbert)

directly from the network corresponding to the size of the layer. However, for embedding or attention layers, which are structured to process sequences, an aggregation function must be used to reduce the layer to a simple activation vector. In this example, for layers of more than one dimension, we use the first token of the sequence, as it is also the one used for the classification task (the CLS token) as a one-dimensional activation layer. In the other use case below, higher dimensions will be aggregated into a one-dimensional vector by averaging activation values.

### 3.3 Using self-organizing maps

An issue with the (naive) method above is that each unit, each neuron, might only participate in the definition of a feature and not represent a feature itself. Methods based on self-organizing maps (SOMs) have been proposed [6, 12] as a way to identify “prototypical” activation vectors that could be representatives of specific subsets of the data for which a feature is present (in other words, concepts).

A SOM is a neural network that usually consists of a grid in two dimensions (most often a square), where each unit is connected to all units in the input layer and is associated with an activation function corresponding to a distance function between the weight vector of the unit and the input vector. A SOM is not trained in the usual way through backpropagation but by competitive learning. An intuitive description of this process is that, for each example of the training set, a best matching unit (BMU) is first identified as the one with the lowest activation (highest similarity). From this, the weights of all units are updated using the following formula:

$$W'_i = W_i + \theta(BMU, i, \beta) \cdot \alpha \cdot (X - W_i)$$

where  $W_i$  corresponds to the weights of unit  $i$  in the SOM,  $X$  to the input vector,  $\alpha$  is the learning rate (decreasing over time), and  $\theta$  is the neighborhood function applied between the BMU and the unit  $i$  with radius  $\beta$ . This neighborhood function should have value 1 for the BMU itself and gradually decrease (depending on the decaying radius  $\beta$ ) the farther away the unit  $i$  is from the BMU. Over multiple iterations, this creates a map in which similar data points will have BMUs located close to each other, and dissimilar ones will have BMUs in different locations on the map. It has to be noted that SOMs are, fundamentally, clustering methods and that other such methods could have been considered. However, that they are flexible, neural models that organize the created clusters topologically, by similarity, adds some advantages compared to other clustering techniques.

To apply SOMs for mechanistic interpretability, we therefore train one map for each layer of the model on the activation vectors as extracted according to the method described in the previous section. In our example, relying on the `ksom`<sup>3</sup> python library, we trained SOMs of sizes  $5 \times 5^4$  on the entire painter dataset over 5 epochs (iterations).<sup>5</sup> Figure 1 shows a representation of a SOM trained for the `bert . transformer . layer . 3 . ffn` layer of the painter model using colors to show the similarity between adjacent units

<sup>3</sup><https://pypi.org/project/ksom/>

<sup>4</sup> $\alpha = 5 \times 10^{-3}$  with a decay of  $10^{-6}$  per batch of 128 examples, cosine distance as the activation function, a linear neighborhood function,  $\beta = 4$  (with a decay of  $4 \times 10^{-5}$

<sup>5</sup>the code to train SOMs can be found at <https://github.com/mdaquin/actsom>



**Figure 1: Left: Color-based representation of the weight vectors of the `bert . transformer . layer . 3 . ffn` layer (PCA applied to the weight vectors with 3 components for red, green and blue). Right: Percentage of the dataset for which each unit is the BMU (lighter units have higher values).**

and shades to represent the number of samples for which each unit is BMU after training.

### 3.4 Using sparse-autoencoders

Another way to consider the issue mentioned at the beginning of the previous section is expressed in [10] under the term *superposition*: Units of the network do not express *monosemanticity*. That is, they do not correspond to unique interpretable features, but each contributes to the computation of different features. Dictionary learning through sparse autoencoders (SAEs) has been proposed, in particular in [10, 17], as a way to disentangle those contributions to different features. Indeed, SAE models are trained to reproduce the input in the output (i.e., they are autoencoders) but going through an intermediary encoding space of higher dimension than the input/output space and that is made deliberately sparse (i.e., only a few units have high activations at a time). More precisely, in this work, an SAE is a neural network model in which the output is a simple (identity) feedforward layer of the same size as the input. There is one hidden (encoding) feedforward layer with sigmoid activation, which we choose to make three times larger than the input.

Although the architecture of the SAE is mostly straightforward, the key lies in the loss function used to train it through backpropagation. It has two components: a reconstruction loss, measuring how close the output is to the input, and a sparsity loss, measuring how well we have reached the desired level of sparsity in the encoding layer. The first is usually (and in our case) the commonly used mean square error (MSE), which happens to correspond to a Euclidean distance between the input and output. The second is often (and in our case) based on the Kullback–Leibler (KL) divergence between the desired level of sparsity,  $\rho$ , and the achieved level of sparsity,  $\hat{\rho}$ . More precisely, the loss function used to train the SAE is defined as:

$$L(X, \hat{X}) = \frac{1}{2} \|X - \hat{X}\|^2 + \beta \sum_{j=1}^s KL(\rho \| \hat{\rho}_j)$$

where  $X$  is the input of the model,  $\hat{X}$  is the output,  $\beta$  is a regularization parameter that sets the importance of the sparsity loss against the reconstruction loss,  $s$  is the number of units in the encoding layer (the hidden layer) of the SAE,  $\rho$  is the desired sparsity (generally close to 0) and  $\hat{\rho}_j$  is the activation of the unit  $j$  in the encoding layer of the SAE.

As for SOMs, to apply SAE for interpretability, we train them on the activation vectors resulting from the baseline method of Section 3.2. We therefore obtain large encoding vectors of (expected) interpretable features corresponding to three times the size of each of the activation vectors from the baseline method.<sup>6</sup>

## 4 Aligning Interpretable Units to Knowledge Graphs with Decision Trees

Each of the methods described in the previous section, including the baseline, produces a vector of units for each layer of the considered model and data point given as input to that model.<sup>7</sup> For a given vector, each unit is expected to represent a potentially interpretable feature, that is, the value of the unit is expected to reflect the presence or absence of the corresponding feature in the data point. The key question we are trying to address here is: How to verify that the extracted feature is indeed interpretable, and what could be its interpretation?

### 4.1 Description of the process

As mentioned earlier, in Section 2, this is often achieved (at least partially) manually. In [17] for example, the input texts showing the  $n$  highest and lowest activation values for each unit are inspected to understand their differences, i.e. what could be the common characteristics that are present in the high activation examples, and not in the low activation ones.

Our approach follows a similar process, but relies on the assumption that each data point can be connected to an entity in a knowledge graph to try to automate the process, using decision trees to identify the properties of knowledge graph entities that are most representative of the high and low activation examples for a given interpretable unit. In other words, for a given unit  $U$  of an activation vector obtained from one of the three tested methods, the process<sup>8</sup> is as follows:

**Create the activation dataset:** The dataset used for training and validation of the model is applied to the model and to the methods from which  $U$  was obtained to extract, for all data points  $i$ , the activation value  $A(U, i)$  of the unit  $U$ . With  $NA$  the desired size, the dataset is formed from the indexes  $i$  of the  $NA/2$  data points with the highest  $A(U, i)$  and the  $NA/2$  data points with the lowest  $A(U, i)$ . Information on whether a given data point led to a high or a low activation for  $U$  is recorded as a binary target variable (0 for low, 1 for high) for the training of the decision tree in the third step below.

<sup>6</sup>the code to train SAEs is available at [https://github.com/liudmylaklochko/Interpretable\\_MatBench](https://github.com/liudmylaklochko/Interpretable_MatBench), we used  $\beta = 10^{-2}$ ,  $\rho = 10^{-3}$ , a learning rate of  $10^{-2}$  and 1,000 epochs.

<sup>7</sup>The SOM corresponds to a matrix (5x5), but it is flattened to a vector (of size 25 here) to make the process homogeneous between methods.

<sup>8</sup>which implementation can be found at <https://github.com/mdaquin/KG-DeTrActI>

**Obtaining properties from the knowledge graph:** The results from the previous step provides a dataset of indexes of data points in input to the model, and a binary indication on whether they are strongly represented by the candidate interpretable unit  $U$ . The goal of this step is to find descriptors of those data points that could be used to explain this classification. Since, as mentioned above, we make the assumption that every data point is associated with an entity in the knowledge graph (i.e. we can find the identifier, the URI, of the entity in that knowledge graph), we obtain such descriptors by collecting values of properties of those entities at a given number of hops  $NH$ . In other words, if a given index is connected to an entity  $uri_1$ ,  $NH$  is greater than 2 and the triples  $(uri_1, p_1, uri_2)$  and  $(uri_2, p_2, v)$  are present in the knowledge graph, the attribute-value pair  $p_1 : p_2 = v$  will be added in the dataset for the corresponding index. The created dataset, made up for each unit of a set of attribute-value pairs from the knowledge graph and a target binary value for each high or low  $U$ -activation data point, is then post-processed by removing any attributes with more than 50% of missing value, one-hot-encoding categorical values and aggregating by mean multiple values of numerical attributes for a given data point. The remaining missing values are arbitrarily replaced by 0.

**Training decision trees:** Based on the previous steps, for the considered unit  $U$ , we now have a dataset of numerical descriptors (with values in 0, 1 if the original property was categorical) and a target binary variable corresponding to high or low activations of  $U$ . Our goal is to measure the interpretability of  $U$  with respect to the knowledge graph under consideration. In other words, we want to check how much (and which of) the properties of the knowledge graph can be used to explain the high or low activation of  $U$ . To achieve that, we choose to rely on shallow decision trees. A decision tree here is a machine learning model based on a tree of conditions on the numerical values of input descriptors. A shallow decision tree (i.e., one with a low depth) having high accuracy is therefore a model that uses only a few input descriptors to predict the value of the target variable. For the given unit  $U$ , we therefore use the scikit-learn<sup>9</sup> library to build a decision tree of a given maximal depth  $MD$  (all other parameters are left to their default values) on the datasets created in the two previous steps. The accuracy of the created decision tree gives us a measure of how much the knowledge graph properties can be used to interpret the corresponding feature. In addition, since the decision tree construction algorithm involves selecting both the most relevant descriptors and the most relevant thresholds for the conditions on each descriptor, it will include the (small, if  $MD$  is small) subset of knowledge graph properties that are most suitable to characterize  $U$ , together with their value boundaries.

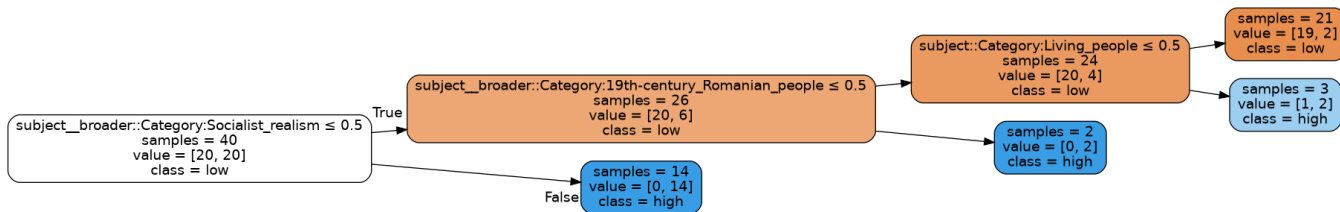


Figure 2: Depth 3 decision tree for unit 24 of the SOM for layer bert . transformer . layer . 3 . ffn.

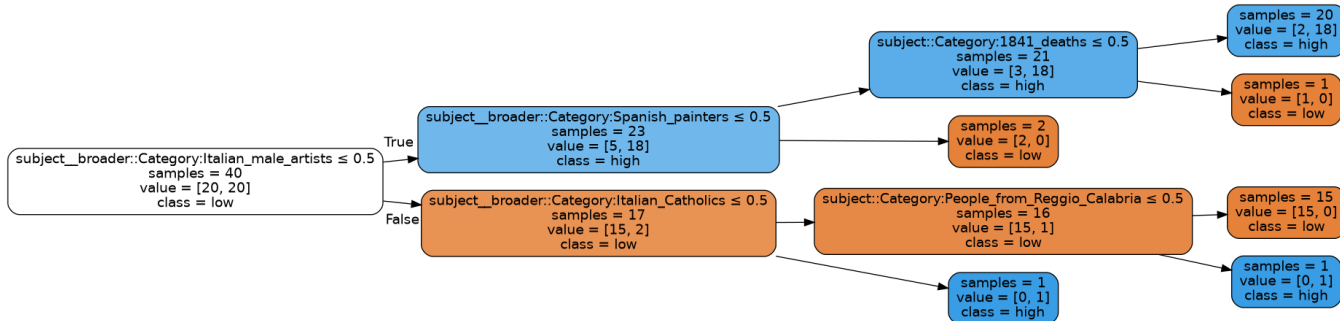


Figure 3: Depth 3 decision tree for unit 10 of the SOM for layer bert . transformer . layer . 5 . ffn.

## 4.2 Illustration on the painter example

As mentioned previously, the painter model used to illustrate the method is based on the biographies of painters extracted from DBpedia. Therefore, each data point in the dataset is naturally associated with the URI of the corresponding painter in the DBpedia knowledge graph. We used the methods described above to produce decision trees of maximum depth  $MD = 3$ , with a number of examples for each unit interpreted  $NA = 40$  (so 20 high and 20 low activation examples) and a number of hops to obtain input descriptors  $NH = 2$ . Since this model is relatively simple (and biased), we obtained in this way decision trees of varying accuracies for units extracted using the three methods described above. For example, the decision tree in Figure 2 was obtained from unit 24 of the SOM for a given layer of the model, and had a precision of 92.5%. The unit seems to primarily identify painters of the socialist realism movement, and for those not belonging to this category, separates further depending on the era (19th century, living people) and localization (Romania). Another example is shown in Figure 3, which shows that, in a later layer of the model, unit 10 seems to be primarily corresponding to opposing Italian (male) painters to Spanish painters, with further details looking at religion, specific regions, and time. This tree obtains 95% accuracy, showing that those descriptors are strongly related to a feature in that layer of the network, as extracted through the SOM method.

## 5 Use case in Material Science

Although the above examples with the painter model give us an indication of the feasibility and potential value of the approach to align candidate interpretable features with knowledge graphs, it remains limited considering the simplicity of the model and the

noise in both the data and the knowledge graph used. In this section, we, therefore, experiment with a more complex and more robust use case in the domain of material science. We apply the two interpretability methods and the baseline on a GNN for the atomic structure of materials with a bespoke knowledge graph and compare the results obtained across selected layers and with different depth thresholds for the decision trees.

### 5.1 Description of the use case

Material science has had an increasing involvement with AI models, especially in relation to material discovery and material design. Indeed, traditional methods for computing some physical properties of materials (which might or might not have already been synthesized) are either too imprecise or extremely resource-intensive. For this reason, many deep learning models have appeared that aim to provide fast and accurate predictions. MEGNet [3] is one of those models that have been shown to achieve good results in a number of tasks related to the prediction of various physical properties of materials.<sup>10</sup>

*Model and dataset:* MEGNet is a GNN: It takes a representation of a graph as input, with nodes and edges associated with various features. Here, the graph corresponds to the atomic structure of a material. Each node corresponds to an atom in this structure and the edges to interactions between atoms. The model itself is relatively complex: it includes encoding layers, three blocks carrying out graph convolution operations, followed by an output block itself made up of multiple layers.

<sup>10</sup>See results published on the MatBench leader board at [https://matbench.materialsproject.org/Full%20Benchmark%20Data/matbench\\_v0.1\\_MegNet\\_kgcnn\\_v2.1.0/](https://matbench.materialsproject.org/Full%20Benchmark%20Data/matbench_v0.1_MegNet_kgcnn_v2.1.0/).

<sup>9</sup><https://scikit-learn.org/>

We chose to rely on a pretrained version of MEGNet for the task of predicting the formation energy of a material, which corresponds to the change in energy of a material when formed from its constituent atoms compared to the energy of those atoms in the reference states of the corresponding elements. This pretrained version is provided through the `matgl`<sup>11</sup> library, which also provided us with the reference dataset for this pretrained model (close to 70K materials), including material structures and formation energy as computed by the Material Database Project<sup>12</sup> (MDP). Each material in the dataset is associated with a unique identifier within MDP.

*Knowledge graph:* As part of a related project, a knowledge graph is currently being built regarding materials and their properties, from multiple sources, including MDP as well as the Mendeleev library. We reuse this knowledge graph here, where each material is identified by a URI constructed from its identifier in MDP. It is associated through various properties with physical characteristics such as its size in number of atoms, its formula, the chemical elements it includes, etc. Each chemical element is also identified as an entity and associated with physical or chemical properties from Mendeleev, including the group, period, electronegativities, etc.

## 5.2 Application of the interpretability methods

We applied the two methods presented in Section 3 and the baseline to the MEGNet model using the entire formation energy dataset.<sup>13</sup> For the baseline method, we aggregated activation vectors of layers of more than 1 dimension by the mean over the relevant dimension. The details of the parameters used for the training of the SOMs and SAEs are those given in Section 3.

## 5.3 Results of alignment with the knowledge graph

We applied the approach presented in Section 4.1 using the knowledge graph described shortly in the previous section on a selected subset of 7 layers in MEGNet, taken from different levels of the models (from bottom to top). The created datasets each include 40 materials with high and low activations for each unit ( $NA = 40$ ) and we set the number of hops to obtain information to 2 ( $NH = 2$ ). We varied the maximum depth  $MD$  of the decision trees from 1 to 5 to measure the evolution of the precision of the decision trees according to this parameter, in the expectation that more detailed interpretations (deeper decision trees) would lead to higher precisions.

Figure 4 shows the average precision for each method depending on the maximum depth threshold applied when building the decision tree. Since each method produces a different number of units and not all units of the methods are expected to be interpretable (i.e., some are redundant), we show the average over the top 175 decision trees for each method at each depth threshold.<sup>14</sup> Naturally, the precision of the decision trees tends to increase with

the depth threshold. It could also be noted that this accuracy is already high even at low depth, showing that the mechanistic interpretation methods used (SOM and SAE) extract features that tend to be interpretable, even with a limited knowledge graph. Finally, as expected, the worst performing method, especially at low depth, is the baseline method, which confirms some of the assumptions on which the two other methods are based. In other words, even if original activations are already, to an extent, interpretable (at least more than random). As can be seen, given the filters applied, SOM and SAE appear to achieve similar results, although as will be seen later, not in exactly the same way. It can be noticed as well that there are greater difficulties in training robust SAEs compared to SOMs. In other words, further fine-tuning of the hyperparameters of the SAEs might lead to even greater results in interpretability performances.

Table 1 shows the average accuracy obtained by the baseline and the two interpretability methods in each of the layers, with  $MD = 1$ , sampling the 16 best decision trees for each method and each layer<sup>15</sup>. The order of the layers depends on the order in which they appear in the model (earlier layers first). As can be seen, the trend per layer tends to follow the general trend: Even very shallow decision trees (of depth 1) are able to create conditions that, on average, reflect the identified features with high accuracy. Also, as in Figure 4, SOM and SAE achieve similar performances for all layers, with only small differences. The baseline method is always the worst-performing method, even if there seems to be a correlation between the performance of the baseline and that of SAE. With exceptions, the results tend to be similar from one layer to another for a given method, although *blocks.0.activation* (the earliest, in the model, of the tested layers) and *out\_put\_proj.layers.2* (the latest of the tested layers) both achieve lower accuracies with all three methods.

**Table 1: Average precision of the top 16 decision trees per method, per layer.**

	baseline	SOM	SAE
<i>blocks.0.activation</i>	0.743	0.815	0.780
<i>blocks.1.conv_node_func.layers.3</i>	0.743	0.847	0.795
<i>blocks.1.state_func.layers.0</i>	0.814	0.868	0.820
<i>blocks.2.conv_node_func.layers.3</i>	0.743	0.858	0.796
<i>blocks.2.state_func.layers.0</i>	0.809	0.855	0.835
<i>out_put_proj.layers.0</i>	0.778	0.863	0.814
<i>out_put_proj.layers.2</i>	0.744	0.779	0.788

Finally, in Table 2, we show the (up to) three properties most used at the first level of decision trees built for each of the three methods, for each of the selected layers (or, in other words, the properties used in trees of depth 1). This is interesting as it enables us to consider the kind of interpretable features each method tends to build and to determine whether there are significant differences between them. As can be seen, although some properties tend to recur across all three methods, there does not seem to be a clear

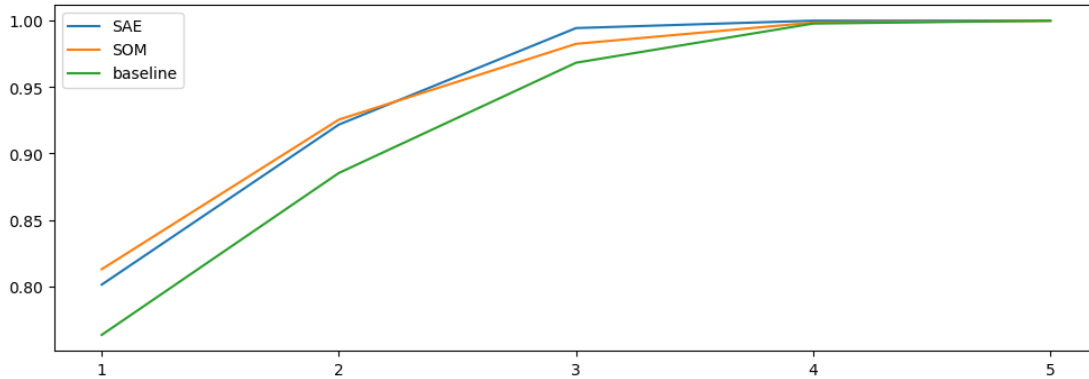
<sup>15</sup>since the smallest activation vector across all methods is 16 (baseline).

<sup>11</sup><https://github.com/materialsvirtuallab/matgl>

<sup>12</sup><https://next-gen.materialsproject.org/>

<sup>13</sup>the code to extract activations from MEGNET can be found at [https://github.com/liudmylaklochko/activations\\_MEGNet](https://github.com/liudmylaklochko/activations_MEGNet)

<sup>14</sup>175 is chosen as the smallest number of decision trees generated for any method (SOM) over the 7 selected layers.



**Figure 4: Evolution of the average precision of the top 175 decision trees for the three methods across units of all selected layers.**

pattern in the kind of material descriptors with which features extracted by the methods are aligned. Among the common features, especially extracted by the SOM method, we can, for example, mention those related to the size of materials (`nsites`) and the complexity of their structure (`nelements`). Another example is the property called “electron affinity” of the included elements in the material, which is especially present in the baseline and SAE methods. This seems relevant considering that electron affinity is defined as the energy change that results from adding an electron to a gaseous atom, which is intuitively related to formation energy (the quantity being predicted by the model). It can actually be noticed that baseline and SAE appear to have more overlap with each other than with SOM. This might be due to the fact that the SAE-based encoding is really a higher-dimensional representation of the information contained in the activation vectors interpreted in the baseline method, while the SOM units represent prototypical values of those activation vectors akin to the centroids of clusters. SAEs are therefore much closer in nature to the original activation vectors than SOM. We could hypothesize, therefore, that SAEs here obtain more interpretable features than the baseline through denoising and decorrelating the latent features already present there, while SOM achieves interpretability by abstracting this latent space into a different, more general one.

## 6 Conclusion

In this paper, we have proposed a method for using knowledge graphs as a way to automatically measure the interpretability of and characterize candidate interpretable features obtained from two mechanistic interpretability methods (as compared to a baseline corresponding to the base activation vectors of the interpreted neural network). Under the assumption that data points are associated to knowledge graph entities, the approach automatically relates knowledge graph properties to those interpretable features by training decision trees to recognize high or low activations for those features based on those properties. In doing so, we not only obtain, through the precision of the decision tree, a measure of the interpretability of those features with respect to the knowledge

graph, but also a characterization of those features using the vocabulary of the knowledge graph.

Through a complex use case in material science, we showed the feasibility of the approach in a realistic setting and that it enables the identification of differences in the performance and characterization of different methods. We, in particular, have shown that both methods could identify features that strongly align with properties of the considered knowledge graph better than the baseline, but with notable differences in both the precision of the alignment and the properties to which they aligned.

There are naturally a number of aspects on which further work could be done with respect to this approach. Firstly, different choices could be made on the interpretability methods to test, as well as on the way to measure interpretability (using other models than decision trees and other measures than accuracy). On a practical level, the scalability of the technique itself could be improved. Indeed, computing the interpretability methods requires collecting a large number of potentially very large activation vectors, which then all have to be related to properties obtained from traversing the knowledge graph for potentially several hops. Engineering challenges appear in processing those activation vectors, storing them, and reliably querying knowledge graphs to form the activation datasets used here. Although the presented use case, which is a relatively complex deep learning model, shows that this is feasible, setting up a generic, robust, and scalable framework to apply this to even larger models and even larger datasets remains to be done.

Also, it is an interesting feature of this approach that the results are dependent not only on the quality of the model being tested, but also on the quality of the knowledge graph used for its interpretation. There are many domains (especially scientific domains) in which deep learning models are being used, while strong foundational knowledge also exists. The approach presented here enables measuring and characterizing interpretability *with respect to a knowledge graph*. This means in particular that it can be applied only under the condition that the foundational knowledge mentioned above is first encoded into such a knowledge graph. Assuming this is done, applying our approach could enable validating

**Table 2: Three most used properties in decision trees of depth 1 for each layer and each method.**

	baseline	SOM	SAE
blocks.0.activation	includesElement:electron_affinity, includesElement:abundance_crust, includesElement:c6_gb	nelements, formation_energy_per_atom, total_magnetization_normalized_formula_units	includesElement:atomic_volume, includesElement:electron_affinity, includesElement:proton_affinity
blocks.1.state_func.layers.0	includesElement:series_id, includesElement:atomic_radius_rahm, includesElement:electronegativity_allen	nelements, density_atomic, includesElement:abundance_sea	includesElement:series_id, includesElement:c6, total_magnetization
blocks.1.conv.node_func.layers.3	includesElement:abundance_crust, includesElement:electron_affinity, includesElement:metallic_radius	total_magnetization_normalized_vol, theoretical, includesElement:specific_heat_capacity	includesElement:molar_heat_capacity, includesElement:c6_gb, includesElement:electron_affinity
blocks.2.state_func.layers.0	includesElement:series_id, includesElement:electron_affinity, energy_above_hull	nelements, total_magnetization, includesElement:proton_affinity	nelements, nsites, includesElement:series_id
blocks.2.conv.node_func.layers.3	includesElement:electron_affinity, includesElement:abundance_crust, cbm	includesElement:abundance_sea, total_magnetization_normalized_vol	includesElement:electron_affinity, includesElement:atomic_weight_uncertainty, includesElement:atomic_volume
output_proj.layers.0	includesElement:lattice_constant, includesElement:covalent_radius_cordero, energy_per_atom	nelements, includesElement:vdw_radius_uff, includesElement:fusion_heat	includesElement:molar_heat_capacity, total_magnetization_normalized_vol, includesElement:group_id
output_proj.layers.2	includesElement:electron_affinity, includesElement:abundance_crust, includesElement:metallic_radius_c12	includesElement:atomic_volume, energy_above_hull, includesElement:molar_heat_capacity	includesElement:electronegativity_gordy, includesElement:vdw_radius_rt, vbm

and debugging deep learning models, and testing how they relate to the practices in the field. However, the analysis of failures of the approach could be the most valuable aspect. Indeed, having found candidate interpretable features that are not well characterized by a given knowledge graph could indicate a failure in the interpretability method or an incompleteness of the knowledge graph. The latter case could provide a starting point for the introduction and formalization of further knowledge in the domain.

## References

- [1] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2017. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6541–6549.
- [2] Leonard Bereska and Efstratios Gavves. 2024. Mechanistic Interpretability for AI Safety—A Review. *arXiv preprint arXiv:2404.14082* (2024).
- [3] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. 2019. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chemistry of Materials* 31, 9 (April 2019), 3564–3572. doi:10.1021/acs.chemmater.9b01294
- [4] Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems* 36 (2023), 16318–16352.
- [5] Abhilekha Dalal, Rushruk Rayan, Adrita Barua, Eugene Y Vasserman, Md Kamruzzaman Sarker, and Pascal Hitzler. 2024. On the value of labeled data and symbolic methods for hidden neuron activation analysis. In *International Conference on Neural-Symbolic Learning and Reasoning*. Springer, 109–131.
- [6] Mathieu d'Aquin. 2023. Finding Concept Representations in Neural Networks with Self-Organizing Maps. In *Proceedings of the 12th Knowledge Capture Conference 2023*. 53–60.
- [7] Maximilian Dreyer, Jim Berend, Tobias Labarta, Johanna Vielhaben, Thomas Wiegand, Sebastian Lopuschkin, and Wojciech Samek. 2025. Mechanistic understanding and validation of large AI models with SemanticLens. *Nature Machine Intelligence* (2025), 1–14.
- [8] Mathieu d'Aquin. 2025. On the role of knowledge graphs in AI-based scientific discovery. *Journal of Web Semantics* 84 (2025), 100854.
- [9] Ruth Fong and Andrea Vedaldi. 2018. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8730–8738.
- [10] Adam S Jermyn, Nicholas Schiefer, and Evan Hubinger. 2022. Engineering monosemanticity in toy models. *arXiv preprint arXiv:2211.09169* (2022).
- [11] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*. PMLR, 2668–2677.
- [12] Valerie Krug, Raihan Kabir Ratul, Christopher Olson, and Sebastian Stober. 2023. Visualizing deep neural networks with topographic activation maps. In *HHAI 2023: Augmenting Human Intellect*. IOS Press, 138–152.
- [13] Jiawei Lu, Qiong Wang, Zhuxiu Zhang, Jihai Tang, Mifen Cui, Xian Chen, Qing Liu, Zhaoyang Fei, and Xu Qiao. 2021. Surrogate modeling-based multi-objective optimization for the integrated distillation processes. *Chemical Engineering and Processing-Process Intensification* 159 (2021), 108224.
- [14] Andriy Nikolov and Mathieu d'Aquin. 2020. Uncovering semantic bias in neural network models using a knowledge graph. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 1175–1184.
- [15] Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. 2024. A Practical Review of Mechanistic Interpretability for Transformer-Based Language Models. doi:10.48550/arXiv.2407.02646 arXiv:2407.02646 [cs].
- [16] Tilman Rauker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. 2023. Toward Transparent AI: A Survey on Interpreting the Inner Structures of Deep

- Neural Networks. doi:10.48550/arXiv.2207.13243
- [17] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, et al. 2024. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. Transformer Circuits Thread.
- [18] Ilaria Tiddi, Mathieu d'Aquin, and Enrico Motta. 2014. Dedalo: Looking for clusters explanations in a labyrinth of linked data. In *The Semantic Web: Trends and Challenges: 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings 11*. Springer, 333–348.
- [19] Yuren Yang, Ye Yuan, Zhen Han, and Gang Liu. 2022. Interpretability analysis for thermal sensation machine learning models: An exploration based on the SHAP approach. *Indoor air* 32, 2 (2022), e12984.
- [20] Yu Zhang, Peter Tiño, Aleš Leonardiš, and Ke Tang. 2021. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5, 5 (2021), 726–742.
- [21] Roland S Zimmermann, David Klindt, and Wieland Brendel. 2024. Measuring per-unit interpretability at scale without humans. *Advances in Neural Information Processing Systems* 37 (2024), 48448–48483.