



KERMITviz: Visualizing Neural Network Activations on Syntactic Trees

Leonardo Ranaldi¹ , Francesca Fallucchi¹ , Andrea Santilli² ,
and Fabio Massimo Zanzotto³ 

¹ Department of Innovation and Information Engineering, Guglielmo Marconi University, Rome, Italy

{l.ranaldi, f.fallucchi}@unimarconi.it

² Department of Computer Science, Sapienza University Rome, Rome, Italy

santilli@di.uniroma1.it

³ Department of Enterprise Engineering, University of Rome Tor Vergata, Rome, Italy

fabio.massimo.zanzotto@uniroma2.it

Abstract. The study of symbolic syntactic interpretations has been the cornerstone of natural language understanding for many years.

Today, modern artificial neural networks are widely searched to assess their syntactic ability, through several probing tasks.

In this paper, we propose a neural network system that explicitly includes syntactic interpretations: Kernel-inspired Encoder with Recursive Mechanism for Interpretable Trees *Visualizer* (KERMITviz). The most important result is that KERMITviz allows to visualize how syntax is used in inference. This system can be used in combination with transformer architectures like BERT, XLNet and clarifies the use of symbolic syntactic interpretations in specific neural networks making the black-box neural network neural networks explainable, interpretable and clear.

Keywords: Natural Language Processing · Explainable AI · Neural Networks

1 Introduction

While systems based on natural language processing (NLP) and neural network (NN) are achieving extraordinary success, the lack of *interpretation* and NN transparency from the representations learned to the underlying decision-making process, is an important problem to be addressed.

Understanding why a model does not correctly classify test data instances or performs incorrectly is a challenging task. Many works propose techniques such as data augmentation [15] or analysis of available features [10] to improve results. Despite the good results that can be obtained on these challenging tasks, usually methodologies do not consider the investigation of the reasons why the model made wrong predictions.

In fact, when human uses an application, that is based on learning to make critical decisions, not having the perception of what is going on, it calls into question the model's *reliability*. To address this problem, researchers have introduced many different techniques to help interpret what is happening in the NNs. Techniques range from

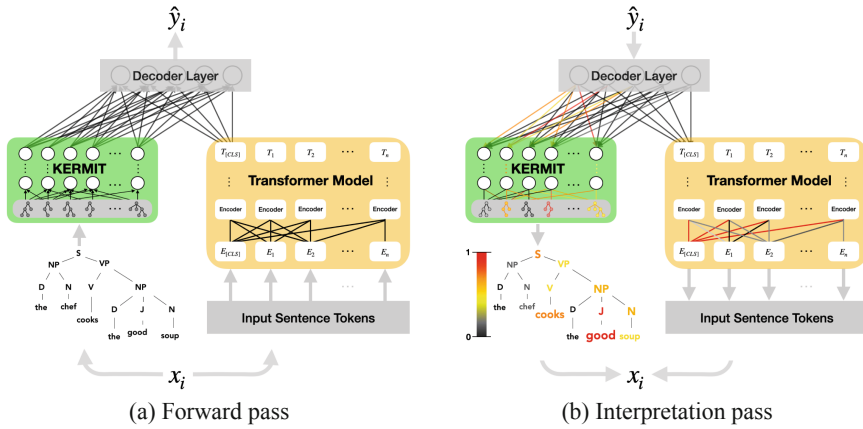


Fig. 1. The KERMIT+Transformer architecture. During the interpretation pass KERMITviz is used to produce *heat parse trees*, while a transformer’s activation visualizer is used for the remainder of the network.

features important explanations [2, 14, 16] to prototypes and criticisms [1, 3]. Feature Importance Explanations methods, given input with features, explain model’s decision by assigning a score to each feature that indicates its contribution in the decision making. Prototype methods seek a minimal subset of samples that can use as condensed view of a data set. In the NLP world, we have word importance explanations. Feature Importance techniques can be categorized into three categories: perturbation-based techniques [14], gradient-based techniques [16], and decomposition-based techniques [2]. However, such techniques are meaningless, specially in the case of NLP, if they are not surrounded by a good method of visualisation that is simple and comprehensible to humans. Many works use these techniques to generate static images, such as *attention maps* [11, 19, 21] and *heat maps* [9] for image classification, indicating which parts of an image are most important for classification. Interaction has also been incorporated into the process of understanding the model through *visual analytics* tools. For example, *ActiVis* [9] offers a view of neuron activations and can be used to view interactive model interpretations of large heterogeneous data formats such as images and text. However *ActiVis* doesn’t support recurrent architectures, a common type of architecture in natural language tasks. For this extent, Ming et al. [11] and Strobel et al. [19] proposed respectively dedicated visualizers for *recurrent neural networks* (*RNNviz*) and *long short-term memory networks* (*LSTMviz*) that are able to inspect the dynamic of the hidden state. These systems are very high performance, provide a very good explanation of what happened and are aimed at both a programmer and an ordinary user, so they are user-friendly. Both *RNNviz* and *LSTMviz* unfortunately do not support specialised RNN-based models, such as memory networks or attention models.

Recently, with the advent of transformer models [20], a lot of work has been done in order to interpret activations of attention heads [7, 21, 22]. All these transformer visualizers allow to view the magnitude of softmax attention heads correlated with input tokens to interpret model’s decisions. By the way of example we selected *BERTviz*

[21] as the representative for this category of Transformer visualizers, it is very difficult to use and its outputs are quite difficult to interpret if you are not familiar with the underlying model. A solution to the problem arising from the difficulty of the task has been proposed by [17,23] that have exploited the basic structure of Transformer to add symbolic syntactic information. Although the symbolic syntactic information is clearer and has allowed good results to be obtained in the downstream tasks, it has not proved useful in terms of explainability. Finally, *Embedding Projector* [18] is an interactive tool for visualizing and interpreting *embeddings*. This tool uses different dimensionality reduction techniques to map high-dimensional embedding vectors into low-dimensional output vectors that are easier to visualize. It can be used to analyze the geometry of words and explore the *embedding space*, although it can't be used directly to explain a neural network model.

In this paper, we present KERMITviz (Kernel-inspired Encoder with Recursive Mechanism for Interpretable Trees *Visualizer*), which is integrated into the KERMIT system [25]. KERMITviz allows researchers to embed symbolic syntactic parse trees into artificial neural networks and to visualize how syntax is used in inference. We use the Layer-wise Relevance Propagation (LRP) [2] (Sect. 2), which is a technique of Feature Importance Explanations. Along the interpretation pass (Fig. 1), using LRP combined with special visualization algorithms, we provide an easy and user-friendly tool to see how syntax is used in inference.

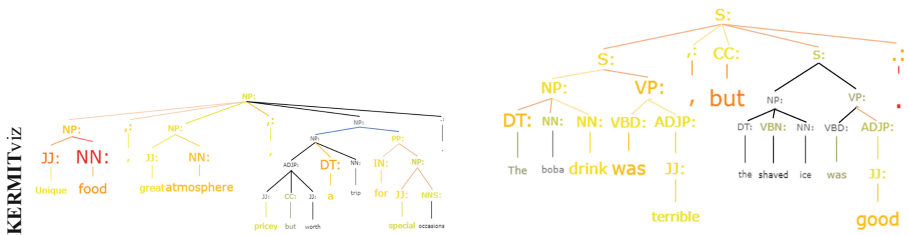


Fig. 2. KERMITviz interpretations over KERMITviz using BERT on two sample sentences where the word *but* is correlated or not with the final polarity.

2 System Description

This section introduces our visualizer KERMITviz stemming from KERMIT [25], a lightweight encoder for universal syntactic interpretations that can be used in combination with transformer-based networks such as BERT [5] (Fig. 1). It follows some preliminary notations (Sect. 2.1), a presentation of KERMIT model (Sect. 2.2), an introduction to KERMITviz (Sect. 2.3) and an overview of heat parse trees (Sect. 2.4).

2.1 Preliminary Notation

Parse trees, \mathcal{T} , are core representations in our model. Parse subtrees τ are recursively represented as trees $t = (r, [t_1, \dots, t_k])$ where r is the label representing the root of the tree and $[t_1, \dots, t_k]$ is the list of child trees t_i . Leaves t are represented as trees $t = (r, [])$.

On parse trees \mathcal{T} , our model KERMIT requires the definition of three sets of subtrees: $N(\mathcal{T})$, $\overline{S}(\mathcal{T})$ and $S(\mathcal{T})$. For defining the last two sets we used subtrees defined in [4]: $N(\mathcal{T})$ contains all the complete subtrees of \mathcal{T} and $\overline{S}(\mathcal{T})$ contains all the valid subtrees of $\mathcal{T} = (r, [t_1, \dots, t_k])$. The set $S(\mathcal{T})$ is the union of $\overline{S}(t)$ for all the trees $t \in N(\mathcal{T})$ and it contains the subtrees used during training and inference.

Finally, to build the untrained KERMIT encoder, we use the properties of random vectors drawn from a multivariate Gaussian distribution $\mathbf{v} \sim \mathcal{N}(0, \frac{1}{\sqrt{d}}\mathbb{I})$. We compose these vectors using the shuffled circular convolution $\mathbf{u} \otimes \mathbf{v}$.

These vectors are drawn from a multivariate Gaussian distribution which guarantees that $(\mathbf{u} \otimes \mathbf{v})^T \mathbf{u} \approx 0$, $(\mathbf{u} \otimes \mathbf{v})^T \mathbf{v} \approx 0$ and $(\mathbf{u} \otimes \mathbf{v}) \neq (\mathbf{v} \otimes \mathbf{u})$. This operation is a circular convolution \star (as for Holographic Reduced Representations [13]) with a permutation matrix Φ : $\mathbf{u} \otimes \mathbf{v} = \mathbf{u} \star \Phi \mathbf{v}$.

2.2 The Encoder for Exploiting Parse Trees and Sub-network

KERMIT is a neural network that allows to encode and directly use syntactic interpretations in neural networks architectures. The KERMIT neural network has two main components: the KERMIT encoder, that encodes parse trees \mathcal{T} in embedding vectors, and a multi-layer perceptron (MLP) that exploits these embedding vectors:

$$\mathbf{y} = \mathcal{D}(\mathcal{T}) = \mathbf{W}_{dt} \mathbf{x} \quad (1)$$

$$\mathbf{z} = \text{mlp}(\mathbf{y}) \quad (2)$$

The KERMIT encoder \mathcal{D} in Eq. 1 stems from tree kernels [4] and distributed tree kernels [24]. It gives the possibility to represent parse trees in vector spaces \mathbb{R}^d that embed huge spaces of subtrees \mathbb{R}^n .

These encoders may be seen as linear transformations $\mathbf{W}_{dt} \in \mathbb{R}^{d \times n}$ (similarly to Transformation in [8]). These linear transformations embed vectors $\mathbf{x}^T \in \mathbb{R}^n$ in the space of tree kernels in smaller vectors $\mathbf{y}^T \in \mathbb{R}^d$:

$$\mathbf{y}^T = \mathbf{W}_{dt} \mathbf{x}^T \quad (3)$$

Columns w_i of \mathbf{W}_{dt} encode subtree $\tau^{(i)}$ and are computed with an encoding function $w_i = \Upsilon(\tau^{(i)})$ as follows:

$$\Upsilon(t) = \begin{cases} r & \text{if } \tau^{(i)} = (r, []) \\ r \otimes \Upsilon(\tau^{(i)}) \otimes \dots \otimes \Upsilon(\tau^{(k)}) & \text{if } t = (r, [\tau_1^{(i)}, \dots, \tau_k^{(i)}]) \end{cases}$$

As for tree kernels also for distributed tree encoders, linear transformations \mathbf{W}_{dt} , vectors $\mathbf{x}^T \in \mathbb{R}^n$ are never explicitly produced and encoders are implemented as recursive functions [24].

2.3 Heat Parse Trees and Activation

Heat parse trees (HPTs), similarly to “heat trees” in biology [6], are heatmaps over parse trees (see the colored tree in Fig. 1). The underlying representation is an *active tree* \bar{t} , that is a tree where each node $\bar{t} = (r, v_r, [\bar{t}_1, \dots, \bar{t}_k])$ has an activation value $v_r \in \mathbb{R}$ associated. HPTs are graphical visualizations of active trees \bar{t} where colors and sizes of nodes r depend on their activation values v_r .

As we will see in Sect. 3, this module is used with the Transformers-models architecture but we specify that this part is frozen during this pass, because this analysis is purely syntactic. We compute activation value v_r in *active tree* \bar{t} by using Layer-wise Relevance Propagation (LRP) [2].

LRP is a framework to explain the decisions of a generic neural network using local redistribution rules and is able to explain which input features contributed most to the final classification.

In our case, is used as a sort of inverted function of the MLP in Eq. 2,

$$\mathbf{y}_{LRP} = \text{mlp}_{LRP}^{-1}(z). \quad (4)$$

The property in Eq. 1, that enables the activation of each subtree $t \in \mathcal{T}$ to be computed back by transposing the matrix \mathbf{W}_{dt} , that is:

$$\mathbf{x}_{LRP} = \mathbf{W}_{\mathcal{Y}}^T \mathbf{y}_{LRP} \quad (5)$$

To make the computation feasible, \mathbf{W}_{dt}^T is produced on-the-fly for each tree \mathcal{T} . Finally, activation values v_r of nodes $r \in \mathcal{T}$ are computed by summing up values $\mathbf{x}_{LRP}^{(i)}$ if $r \in t^{(i)}$.

2.4 Visualizing Activation in Heat Parse Trees

KERMITviz give the possibility to visualize the activation of parse trees. To make the active trees understandable we use *heat maps*. We use this tool to visualize how much a subtree affects the final decision of an NN classifier Fig. 2. We define the *Heat parse trees* as a graphical visualization of heatmaps over active trees \bar{t} where colors and sizes of nodes r depend on their relevance values v_r . The module allows us to explain which nodes have contributed most to the final classification of a model through the visualization of the Heat parse trees.

3 KERMITviz - System Overview

KERMITviz is a visualizer to inspect how syntax is used in taking final decisions in specific tasks. We showed that KERMIT can effectively embed different syntactic information and KERMITviz can explain KERMIT’s decisions. KERMITviz offers two main features: the visualization tool and a tutorial on how to quickly build and visualize a sample KERMIT encoder network¹.

¹ The code is available at <https://github.com/ART-Group-it/KERMIT>.

KERMIT 🤖

This is a demonstration version of KERMIT on Google Colab

```

1 from kerMIT.tree_encode import parse as parse_tree
from kerMIT.samples import willis
from kerMIT.dtk import DT
from kerMIT.operation import fast_shuffled_convolution
from kerMIT.explain import kerMITviz

#parser definition
calculator = DT(dimension=4096, LAMBDA=0.4, operation=fast_shuffled_convolution)

#sentence in input
sentence = "The chef cooks the good soup"

tree_sentence, dtk_sentence, bert_sentence = utils.get_sentence(sentence, calculator)


2 from kerMIT.explain_pytorch import LRP_linear_layer as LRP_t
#prediction of the input sentence
y_predict = model.get_activation(bert_sentence, dtk_sentence).cpu()
#calculation of contributions through the LRP algorithm
hin, w, b, hout, Rout, bias_nb_units, eps, bias_factor = LRP_t.prepare_input_LRP(y_predict, dtk_sentence, model, BERT_DIM)
Rin = LRP_t.lrp_linear_torch(hin.cpu(), w.cpu(), b.cpu(), hout.cpu(), Rout.cpu(), bias_nb_units, eps, bias_factor, debug=False)
act_lrp = act.ActivationSubtreeLRP(calculator)
act_tree_lrp = act_lrp.activationOC(Rin.detach().numpy(), tree_sentence)
#visualization of previously extracted activations through heat parse tree
heat_parse_tree = kerMITviz.assign_contribution_nodes(act_tree_lrp)

100% ██████████ 1/1 [00:00<00:00, 156.18it/s]

3 IPython.display.HTML(filename='/content/KERMIT/Visualizer/notebook/index2_new.html')

```

Smart visualization of Heat Parse Trees for KERMIT



Download Image

© ART Group, Università degli Studi di Roma "for Vergata", code available on [GitHub](#)

Fig. 3. KERMIT’s notebook with integrated KERMITviz is available on google colab.

In this paper, we focus specifically on these two use cases: an example of the advantages provided by the *heat parse tree* produced by KERMITviz (Fig. 2), and an example on how to generate an *heat parse tree* given a sentence (see Fig. 3).

Hereafter, are described these use cases and demonstrate them in Sect. 3.1 and Sect. 3.2.

3.1 KERMITviz - Example

KERMITviz allows to visualize activation as *heat parse trees* to help justify the choices of the model based on the relevance of the fragments of a sentence, as shown in Fig. 2. In this figure (Fig. 2), colors and sizes identify the relevance value v_r of node r . The range of values: $v_r \in [0, 1]$ and colour range goes from *black* for v_r tending to 0 and *red* for v_r tending to 1.

3.2 KERMITviz - Demo

Interacting and testing KERMITviz is very easy. We produced a jupyter notebook to explain how to get a *heat parse tree* given sentence. We also give users the opportunity to try out this system without the need to download and install everything, using of Google Colaboratory (GC)². In Fig. 3 are represented some important steps of KERMITviz on the GC platform. More precisely, we can observe the encoding of the sentence in the constituent tree, the encoding in vectors introduced in Sect. 2.2, the application of the LRP algorithm in Sect. 2.3 and finally the visualisation using the *heat parse trees* Sect. 2.4.

4 Conclusion

KERMITviz is a simple visualizer that allows us to explain how syntactic information is used in classification decisions within networks combining KERMIT and BERT.

KERMITviz has a clear description of the used syntactic subtrees and gives the possibility of visualizing how syntactic information is exploited during inference, this opens consequently the possibility of devising models to include explicit syntactic inference rules in the training process.

Our future goal is to combine KERMITviz with a *rule control mechanism* of a [12] neural network in order to have full control over the decisions of a fully explainable neural network.

References

1. Ancona, M., Ceolini, E., Öztireli, C., Gross, M.: Towards better understanding of gradient-based attribution methods for deep neural networks (2018)
2. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS ONE **10**(7), 1–46 (2015). <https://doi.org/10.1371/journal.pone.0130140>
3. Bien, J., Tibshirani, R.: Prototype selection for interpretable classification. Ann. Appl. Stat. **5**(4) (2011). <https://doi.org/10.1214/11-aos495>. <http://dx.doi.org/10.1214/11-AOAS495>
4. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In: Proceedings of ACL 2002 (2002)
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.0 (2018). <http://arxiv.org/abs/1810.04805>
6. Foster, Z.S.L., Sharpton, T.J., Grünwald, N.J.: Metacoder: an R package for visualization and manipulation of community taxonomic diversity data. PLoS Comput. Biol. **13**(2) (2017). <https://doi.org/10.1371/journal.pcbi.1005404>
7. Hoover, B., Strobel, H., Gehrmann, S.: exBERT: a visual analysis tool to explore learned representations in transformers models. arXiv e-prints [arXiv:1910.05276](https://arxiv.org/abs/1910.05276), October 2019
8. Johnson, W., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. Contemp. Math. **26**, 189–206 (1984)

² <https://colab.research.google.com>.

9. Kahng, M., Andrews, P.Y., Kalro, A., Chau, D.H.: ActiVis: visual exploration of industry-scale deep neural network models. CoRR abs/1704.01942 (2017). <http://arxiv.org/abs/1704.01942>
10. Liang, B., Yin, R., Gui, L., Du, J., He, Y., Xu, R.: Aspect-invariant sentiment features learning: Adversarial multi-task learning for aspect-based sentiment analysis. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM 2020, New York, NY, USA, pp. 825–834. Association for Computing Machinery (2020). <https://doi.org/10.1145/3340531.3411868>
11. Ming, Y., et al.: Understanding hidden memories of recurrent neural networks. CoRR abs/1710.10777 (2017). <http://arxiv.org/abs/1710.10777>
12. Onorati, D., Tommasino, P., Ranaldi, L., Fallucchi, F., Zanzotto, F.M.: Pat-in-the-loop: declarative knowledge for controlling neural networks. *Future Internet* **12**(12) (2020). <https://doi.org/10.3390/fi12120218>. <https://www.mdpi.com/1999-5903/12/12/218>
13. Plate, T.A.: Holographic reduced representations. *IEEE Trans. Neural Networks* **6**(3), 623–641 (1995). <https://doi.org/10.1109/72.377968>
14. Ribeiro, M.T., Singh, S., Guestrin, C.: “Why should i trust you?": explaining the predictions of any classifier (2016)
15. Rizos, G., Hemker, K., Schuller, B.: Augment to prevent: short-text data augmentation in deep learning for hate-speech classification. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, New York, NY, USA, pp. 991–1000. Association for Computing Machinery (2019). <https://doi.org/10.1145/3357384.3358040>
16. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-CAM: visual explanations from deep networks via gradient-based localization. *Int. J. Comput. Vision* **128**(2), 336–359 (2019). <https://doi.org/10.1007/s11263-019-01228-7>. <http://dx.doi.org/10.1007/s11263-019-01228-7>
17. Shen, Y., Lin, Z., Huang, C.W., Courville, A.: Neural language modeling by jointly learning syntax and lexicon (2018)
18. Smilkov, D., Thorat, N., Nicholson, C., Reif, E., Viégas, F.B., Wattenberg, M.: Embedding projector: interactive visualization and interpretation of embeddings. arXiv preprint [arXiv:1611.05469](https://arxiv.org/abs/1611.05469) (2016)
19. Strobel, H., Gehrmann, S., Huber, B., Pfister, H., Rush, A.M.: Visual analysis of hidden state dynamics in recurrent neural networks. CoRR abs/1606.07461 (2016). <http://arxiv.org/abs/1606.07461>
20. Vaswani, A., et al.: Attention is all you need. In: NIPS (2017)
21. Vig, J.: A multiscale visualization of attention in the transformer model. In: ACL 2019–57th Annual Meeting of the Association for Computational Linguistics, Proceedings of System Demonstrations, pp. 37–42 (2019)
22. Wallace, E., Tuyls, J., Wang, J., Subramanian, S., Gardner, M., Singh, S.: AllenNLP Interpret: a framework for explaining predictions of NLP models. In: Empirical Methods in Natural Language Processing (2019)
23. Wang, Y., Lee, H.Y., Chen, Y.N.: Tree transformer: integrating tree structures into self-attention. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, pp. 1061–1070. Association for Computational Linguistics, November 2019. <https://doi.org/10.18653/v1/D19-1098>. <https://www.aclweb.org/anthology/D19-1098>
24. Zanzotto, F.M., Dell’Arciprete, L.: Distributed tree kernels. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, vol. 1, pp. 193–200 (2012). <http://www.scopus.com/inward/record.url?eid=2-s2.0-84867126965&partnerID=MN8TOARS>

25. Zanzotto, F.M., Santilli, A., Ranaldi, L., Onorati, D., Tommasino, P., Fallucchi, F.: KERMIT: complementing transformer architectures with encoders of explicit syntactic interpretations. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 256–267. Association for Computational Linguistics, November 2020. <https://doi.org/10.18653/v1/2020.emnlp-main.18>. <https://www.aclweb.org/anthology/2020.emnlp-main.18>