




Article

Pat-in-the-Loop: Declarative Knowledge for Controlling Neural Networks

Dario Onorati ¹, Pierfrancesco Tommasino ¹, Leonardo Ranaldi ² , Francesca Fallucchi ^{2,*}  and Fabio Massimo Zanzotto ^{1,*} 

¹ Department of Enterprise Engineering, University of Rome Tor Vergata, 00133 Roma, Italy; darionorati@gmail.com (D.O.); tommasinofrancesco93@gmail.com (P.T.)

² Department of Innovation and Information Engineering, Guglielmo Marconi University, 00193 Roma, Italy; l.ranaldi@unimarconi.it

* Correspondence: f.fallucchi@unimarconi.it (F.F.); fabio.massimo.zanzotto@uniroma2.it (F.M.Z.)

Received: 22 October 2020; Accepted: 28 November 2020; Published: 2 December 2020



Abstract: The dazzling success of neural networks over natural language processing systems is imposing an urgent need to control their behavior with simpler, more direct declarative rules. In this paper, we propose Pat-in-the-Loop as a model to control a specific class of syntax-oriented neural networks by adding declarative rules. In Pat-in-the-Loop, *distributed tree encoders* allow to exploit parse trees in neural networks, *heat parse trees* visualize activation of parse trees, and parse subtrees are used as declarative rules in the neural network. Hence, Pat-in-the-Loop is a model to include human control in specific natural language processing (NLP)-neural network (NN) systems that exploit syntactic information, which we will generically call Pat. A pilot study on question classification showed that declarative rules representing human knowledge, injected by Pat, can be effectively used in these neural networks to ensure correctness, relevance, and cost-effective.

Keywords: NLP; machine learning; deep learning; AI; human-in-the-loop

1. Introduction

Neural networks are obtaining dazzling successes in natural language processing (NLP). General neural networks learned on terabytes of data are replacing decades of scientific investigations by showing unprecedented performances in a variety of NLP tasks [1]. Hence, systems based on NLP and on neural networks (NLP-NN) are everywhere.

As a consequence of their success, public opinion is extremely fast in spotting possibly catastrophic, unwanted behavior on deployed NLP-NN systems (see, for example, [2,3]). As many learned systems [4,5], NLP-NN systems are also exposed to biased decisions or biased production of utterances. This problem is becoming so important that extensive analyses are performed, for example, for the tricky class of systems for sentiment analysis [6].

To promptly recover from catastrophic failures, NLP-NN systems should be endowed with the possibility of modifying their behavior by using declarative languages. Deductive teaching is an extremely difficult task even in the human learning process [7,8]. Active learning techniques [9,10] can require too many examples and may focus the attention of NLP-NN systems on irrelevant peculiarities of datasets [11]. Usually we do not have the time or budget for human input on every data point, and so need strategies for deciding which data points are the most important for human review. Due to the high costs to obtain human-generated activity data using solutions for which a very limited number of examples with supervised information such as Few-Shot Learning or One-shot learning [12,13] could be used. But the core issue of these techniques is the unreliable empirical risk minimizer that makes them hard to learn. Understanding the core issue helps categorize different works into data,

model and algorithm according to how they solve the core issue using prior knowledge: data augments the supervised experience, model constrains the hypothesis space to be smaller, and algorithm alters the search strategy for the best hypothesis in the given hypothesis space [14]. But this is exactly what we want to fight in favour of approaches that understanding neural networks and trying to control their behavior besides using training examples.

Looking into NLP-NN systems beyond the dazzling light is becoming an active area [15,16], since traditional neural network visualization tools are obscure when applied to NLP-NN systems. Heatmaps are powerful tools for visualizing neural networks applied to image interpretation [17]. In fact, heatmaps can visualize how neural network layers treat specific subparts of images. Yet, when applied to NLP-NN systems [18] they are extremely difficult to interpret. For this reason, human involvement with the right interfaces could expedite the efficient labeling of tricky or novel data that a machine can't process, reducing the potential for data-related errors.

In this paper, we propose *Pat-in-the-Loop* as a model to include human control in specific NLP-NN systems that exploit syntactic information. The key contributions are: (1) *distributed tree encoders* that directly exploit parse trees in neural networks; (2) *heat parse trees* that visualize which parts of parse trees are responsible for the activation of specific neurons (see Figure 1); and (3) a declarative language for controlling the behavior of neural networks.



Figure 1. A heat parse tree.

Distributed tree encoders allow to produce heat parse trees and developers can explore activation of parse trees for specific decisions to derive rules for correcting system behavior.

In the following work, we performed a pilot study on question classification where *Pat-in-the-Loop* showed that human knowledge can be effectively used to control the behavior of a syntactic NLP-NN system.

In the next section (Section 2) we report the related works about the visualization of neural networks models. Next follow a description of *Pat-in-the-Loop* works (Section 3) and finally (Section 4), we show the improvements achieved by the proposed model.

2. Related Work

In recent years with the advent of neural networks, many methods to visualize neural networks have been developed. The most common methods to display neural networks is using a *node-link graph* where nodes depict computational units and *edge weights* indicate an input-output connection between these nodes. Generally, for ease of understanding and to encourage the user, the magnitude of a parameter or activation is displayed using different colors and sizes for the *edge weights*.

For example, *ActiVis* [19] 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.

ActiVis can closely integrate multiple coordinated views, such as a model architecture calculation graph and a neuron activation view for model discovery and comparison, users can explore complex models of deep neural networks at both instance and subset level. Although it is a progressive

system, *ActiVis* does not support recurrent architectures, a common type of architecture in natural language tasks.

For this extent, Ming et al. [20] and Strobel et al. [21] 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. The ultimate purpose is to show the functions of hidden state units and explain them using their expected response to input texts, i.e., words. This allows users to gain a more complete understanding and greater confidence in the hidden RNN and LSTM mechanism through various visual techniques.

Recently, with the advent of transformer models [22], a lot of work has been done in order to interpret activations of attention heads [23–25]. In this new world of multi-layered, multi-headed attention, mechanisms of the Transformer model can be difficult to decipher. To make the model more accessible, many researchers have begun to think about an open-source tool that visualizes attention at multiple scales, each of which provides a unique perspective on the attention mechanism. All these Transformer visualizers allow to view the magnitude of softmax attention heads correlated with input tokens to interpret model's decisions. By way of example, we selected *BERTviz* [23] as the representative for this category of transformer visualizers.

Embedding Projector [26] 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 cannot be used directly to explain a neural network model.

The following table (Table 1) shows a sample of the most common types of visualization tools for neural networks in the context of natural language processing.

Table 1. Summary of representative visualization tools for natural language processing (NLP)-neural network (NN) systems. * our work.

Features	*	<i>RNNviz</i>	<i>Emb. Proj.</i>	<i>LSTMVis</i>	<i>ActiVis</i>	<i>BERTviz</i>
Interpretability & Explainability	x	x		x	x	x
Debugging & Improvement Models	x			x	x	
Developer-friendly	x	x	x	x	x	x
User-friendly			x	x	x	x
Algorithm Attribution & Features Visualization	x		x			x
During Training						
After Training	x	x	x	x	x	x
NLP-NN system	x	x	x	x	x	x

From Table 1, we can observe the basic characteristics offered by the above mentioned works. The features that everyone shares are: the target audience, i.e., *Developer-Friendly*, the time of training when we can avail ourselves of these systems, i.e., *After Training* and finally the purpose of the systems themselves, i.e., improve the elements *Interpretability & Explainability*. The distinguishing features offered by our system are: the display and easy choice of the underlying model to use, i.e., *Algorithm Attribution & Features Visualization* and the ability to manipulate the model itself to improve it in a very simple way.

In addition to this visualizer, we propose *Pat-in-the-Loop* as a model to include human control in specific NLP-NN systems that exploit syntactic information. Our system allows to display *heat parse trees* that are a handy way to represent syntactic node contributions in a neural network directly

into syntactic trees and a declarative language for controlling the behavior of neural networks. The following section describes in detail how it works.

3. The Model

In Pat-in-the-Loop (see Figure 2), a generic developer, which we call Pat, may inspect the reasons why her/his neural network takes some decisions. In fact, Pat’s neural network model is based on *distributed tree encoders* W_{dt} to directly exploit parse trees in neural networks (Section 3.2). Pat can visualize why some decisions are taken from the network according to parse trees of examples x by using “*heat parse trees*” (Sections 3.1 and 3.3). Hence, Pat can control the behavior of neural networks with declarative rules represented as subtrees by encoding these rules in W_H (Section 3.4).

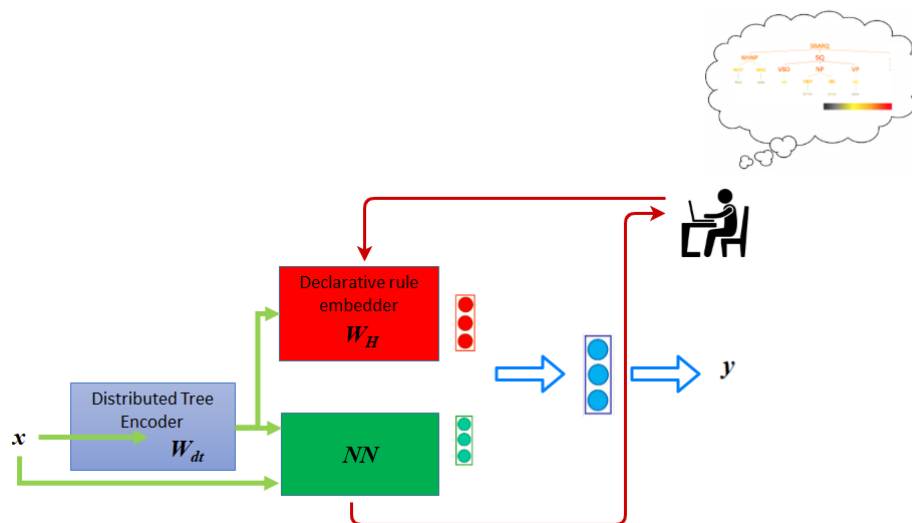


Figure 2. Pat-in-the-Loop: the overall system.

In other words, the key idea we propose in Pat-in-the-Loop model is using “*heat parse trees*” to analyze which parts of parse trees are responsible for the activation of specific neurons (Section 3.3); and, then, controlling the behavior of neural networks with declarative rules derived from the analysis of these heat parse trees (Section 3.4). This is a loop (see the red arrow in Figure 2) where Pat analyzes the output of the Neural Network (NN). The red block, which is the Declarative rule embedder, is a special module that allows Pat to encode declarative rules. These rules, which are embedded in special vectors (see in Section 3.4) will affect the decision of the neural network by modifying its behavior during training.

Before starting the description of the core components of the Pat-in-the-Loop model, Section 3.1 introduces some preliminary notation and the notion of heat parse trees. Below is part relating to the foundations of the proposed system Section 3.2. Then, we close with a section about the visualization (Section 3.3) and the additional layer (Section 3.4).

3.1. Preliminary Notation

Parse trees and *heat parse trees* are core representations in our model. This section introduces the notation to describe these two representations.

Parse trees \mathcal{T} and 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 $t = (r, [])$ with an empty list of children or directly as $t = r$.

Heat parse trees, similarly to “*heat trees*” in biology [27], are heatmaps over parse trees (see Figure 1). The underlying representation is an *active tree* \bar{t} , that is, a tree where an activation value

$v_r \in \mathbb{R}$ is associated to each node: $\bar{t} = (r, v_r, [\bar{t}_1, \dots, \bar{t}_k])$. Heat parse trees are then the graphical visualization of active trees \bar{t} where colors and sizes of nodes r depend on their activation values v_r .

3.2. Distributed Tree Encoders for Exploiting Parse Trees in Neural Networks

Distributed tree encoders are the encoders used in Pat-in-the-Loop to directly exploit parse trees in neural networks. These encoders, stemming from tree kernels [28] and distributed tree kernels [29], give the possibility to represent parse trees in vector spaces \mathbb{R}^d that embed huge spaces of subtrees \mathbb{R}^n .

Tree kernels [28] have offered an important opportunity to fully exploit parse trees in learning with kernel machines [30,31]. Tree kernels are functions implicitly computing the similarity among parse trees \mathcal{T} mapped in vectors $\mathbf{x}^{\mathcal{T}} \in \mathbb{R}^n$ where dimensions are subtrees τ . For example, the 52629-th dimension of $\mathbf{x}^{\mathcal{T}} \in \mathbb{R}^n$ can represent the subtree $\tau^{(52629)} = (SQ, [(VBD, [did]), NP, VP])$ (see Table 2). Vectors $\mathbf{x}^{\mathcal{T}}$ for parse trees T generally have:

$$x_i^{\mathcal{T}} = \begin{cases} \lambda^{\frac{|\tau^{(i)}|}{2}} & \text{if } \tau^{(i)} \in S(T) \\ 0 & \text{if } \tau^{(i)} \notin S(T) \end{cases}$$

where $S(\mathcal{T})$ is the set of valid subtrees of \mathcal{T} , $0 < \lambda < 1$ is a decay factor penalizing large subtrees, and $|\tau^{(i)}|$ is the size of the node set of $\tau^{(i)}$. Valid subtrees $\tau \in S(\mathcal{T})$ in [28] are connected subtrees of \mathcal{T} of at least two nodes and, if τ contains a node c , it should contains all the siblings of the node c in \mathcal{T} . For example, $x_{52629}^{\mathcal{T}_e} = \lambda^{\frac{5}{2}}$ for the parse tree in Figure 1 since $\tau^{(52629)}$ is a valid subtree of \mathcal{T}_e . The power of these tree kernels is that parse trees are never explicitly represented as vectors $\mathbf{x}^{\mathcal{T}}$ but the tree kernel functions implicitly compute their dot product.

Table 2. Sample of a subtree space with activation of the target layer o .

Dim in \mathbb{R}^n	Represented Subtree	Target Output o		
		...	o_{27}	...
...
$\tau^{(52628)}$	$(VP, [VBP, ([NP, [(DT, [a])]), NN])])$...	−0.001	...
$\tau^{(52629)}$	$(SQ, [(VBD, [did]), NP, VP])$...	0.11	...
$\tau^{(52630)}$	$(NP, [DT, (NN, [lottery])])$...	0.0002	...
$\tau^{(52631)}$	$(WHNP, [(WDT, [What]), NNS])$...	0.07	...
...

Distributed tree kernels [29] may transfer the opportunity given by tree kernels [28] within neural networks since distributed tree kernels implicitly embed vectors $\mathbf{x}^{\mathcal{T}} \in \mathbb{R}^n$ into a reduced space \mathbb{R}^d in the context of support vector machines. Distributed tree kernels build on Johnson–Lindenstrauss Transformation [32] and holographic reduced representations (HRR) [33].

Building on distributed tree kernels, we propose *distributed tree encoders* that may be seen as linear transformations $\mathbf{W}_{dt} \in \mathbb{R}^{d \times n}$ (similarly to Johnson–Lindenstrauss Transformation [32]). These linear transformations embed vectors $\mathbf{x}^{\mathcal{T}} \in \mathbb{R}^n$ in the space of tree kernels in smaller vectors $\mathbf{y}^{\mathcal{T}} \in \mathbb{R}^d$:

$$\mathbf{y}^{\mathcal{T}} = \mathbf{W}_{dt} \mathbf{x}^{\mathcal{T}}$$

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

$$E(\tau^{(i)}) = \begin{cases} \mathbf{r} & \text{if } \tau^{(i)} = (r, []) \\ \mathbf{r} \otimes E(\tau_1^{(i)}) \otimes \dots \otimes E(\tau_k^{(i)}) & \text{if } \tau^{(i)} = (r, [\tau_1^{(i)}, \dots, \tau_k^{(i)}]) \end{cases} \quad (1)$$

where the operation $u \otimes v$ is the shuffled circular convolution, that is, a circular convolution \star (as for HRR [33]) with a permutation matrix Φ : $u \otimes v = u \star \Phi v$; and, $r \sim \mathcal{N}(0, \frac{1}{\sqrt{d}}\mathbb{I})$ is drawn from a multivariate gaussian distribution.

As for tree kernels also for distributed tree encoders, linear transformations W_{dt} and vectors $x^T \in \mathbb{R}^n$ are never explicitly produced and encoders are implemented as recursive functions [29].

3.3. Visualizing Activation of Parse Trees

Distributed tree encoders give the possibility of using *heat parse trees* to visualize the activation of parse trees in final decisions or intermediate neuron outputs.

To compute of *active trees* \bar{t} useful to produce *heat parse trees*, a neural network should be sliced at the desired layer. Let NN be the sliced neural network, $x = x^T, x^r$ and o its output:

$$o = \text{NN}(W_{dt}x^T, x^r)$$

where, given an example x , x^T is the vector representing the tree \mathcal{T} in the space of subtrees related to the example x , W_{dt} is the distributed tree encoder, and x^r is the rest of the features associated to x .

Using parse trees \mathcal{T} in neural networks is straightforward with distributed trees. In fact, distributed trees $y^T = W_{dt}x^T$ for parse trees \mathcal{T} may be directly used in neural networks as these distributed trees are vectors.

Our heat parse trees show the overlap of activation of subtrees in $S(\mathcal{T})$ of specific trees \mathcal{T} related to a specific example x in a specific net. This shows how subtrees in $S(\mathcal{T})$ contribute to the final activation o_i , that is, a dimension of o . We believe this is more convenient than representing an extremely large heatmap for the list of subtrees in $S(\mathcal{T})$ and their related value o_i (see Table 2).

The computation of active trees \bar{t} for displaying heat parse trees is the following. The activation weight v_r of each node r represents how much the node is responsible for the activation of the overall syntactic tree for the output of the given neuron o_i . Then, the activation value v_r is computed as follows:

$$v_r = \sum_{\tau \in S(\mathcal{T}) \text{ and } r \in \tau} \text{NN}(W_{dt}\lambda^{\frac{|\tau|}{2}}\tau, x^r)$$

where τ is the one-hot vector in the subtree space that indicates the subtree τ and $r \in \tau$ detects in r is node in τ .

With the above computation of \bar{t} , active subtrees τ for the output o_i of a specific neuron are overlapped in single heat parse trees.

The activation value can be calculated in other ways, for example using Layer-wise Relevance Propagation (LRP) [34]. They compute activation value v_r in *active tree* \bar{t} by using LRP, that 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. This method unfortunately does not allow you to split the network at the desired layer, so it has not been taken into account.

3.4. Human-in-the-Loop Layer

Pat now has an important possibility of understanding why decisions are taken by a specific network and, hence, s/he can define specific rules to control the behavior of the neural network. By looking at the activation of specific neurons for specific examples, Pat can understand why the decision has been made. For example, the heat parse tree in Figure 1 suggests that the subtree $(SQ, [VBD, NP, VP])$ is the more active in generating the decision if this is taken for the output of a neuron that represents a final class.

If Pat aims to correct the behavior of the system for a given output, s/he selects the specific instances, derives some declarative rules and embeds these rules into the network to control its behavior. More specifically, Pat selects a subtree τ and insert $E(\tau)$ as a row in matrix W_H that embeds

declarative rules (see Figure 2). This specific rule will affect the decisions made by the network on the example under review and all similar examples when the neural network is re-trained after rule injection in \mathbf{W}_H .

The actual procedure to build up the matrix \mathbf{W}_H is the following. Let us say that Pat aims to capture k different groups of characteristics s/he assumes to be important to control the behavior of the neural network. For each group i , s/he selects a set S_i of subtrees $\tau^{(i)}$ corresponding to the i -th characteristic. The matrix \mathbf{W}_H is then the following:

$$\mathbf{W}_H = \begin{pmatrix} - & w_1 & - \\ - & w_2 & - \\ & \dots & \\ - & w_k & - \end{pmatrix}$$

where $w_i = \sum_{\tau^{(i)} \in S_i} E(\tau^{(i)})$ and $E(\tau^{(i)})$ is specified in Equation (1).

Hence, the matrix \mathbf{W}_H is the editable component of the overall system and the procedure to build-up the matrix \mathbf{W}_H offers an actionable procedure for allowing external agents, that is, Pats, to interact with this neural network-based system. The matrix \mathbf{W}_H can definitely allow external agents to manipulate the behavior of the neural network by encoding rules capturing characteristics they consider relevant for a specific task.

4. Pilot Experiment

We experimented with Pat-in-the-Loop by using a question classification dataset [35]. This data helps to classify the given Questions into respective categories based on what type of answer it expects such as a numerical answer or a text description or a place or human name, etc. The dataset is extremely well studied and performances systems can achieve are very high also if the dataset is extremely small. Hence, the dataset offer a very intriguing possibility to run a complex experiment where a human in the loop can make the difference in calibrating the overall system.

4.1. Experimental Set-Up

We experimented with the Question Classification dataset [35], which contains 5242 training questions and 500 testing questions. We focused on the *coarse grain* classification problem with 6 target classes: Abbreviation (ABBR), Description (DESC), Entity (ENTY), Human (HUM), Location (LOC), and Numeric (NUM).

The Pat-in-the-Loop (see Figure 2) used in the experiments has the following configuration. Distributed trees $\mathbf{W}_{dt}\mathbf{x}^T$ are encoded in a space \mathbb{R}^d with $d = 4000$. The decaying factor of tree kernels is $\lambda = 0.6$. The module $\text{NN}(\mathbf{W}_{dt}\mathbf{x}^T, \mathbf{x}^r)$ is a multi-layer perceptron that combines two multi-layer perceptrons: $\text{Synt}(\mathbf{W}_{dt}\mathbf{x}^T)$ and $\text{Sem}(\mathbf{x}^r)$. Synt exploit syntactic information and its output is 1800. Sem exploits a Bag-of-Word model of the input with word embedding input of 300 from fastText [36] and output of 180. Synt and Sem are concatenated and feed a multi-layer perceptron with two layers: 100 and 6. Finally, \mathbf{W}_H has an input dimension of $d = 4000$ and an output dimension of 6 where 6 is the number of output categories required in the question classification dataset [35]. In this case, we have opted for \mathbf{W}_H , which encodes 6 different characteristics where each characteristic is linked to an output class. Then, the output of \mathbf{W}_H and the output of $\text{NN}(\mathbf{W}_{dt}\mathbf{x}^T, \mathbf{x}^r)$ are concatenated in a single vector that feeds a final linear layer. We used a ReLU activation function among layers. The last activation function is a softmax. The optimizer is Adam [37]. All experiments were run for 20 epochs in Keras [38]. Finally, we used the CoreNLP constituent-based parser [39] for parsing questions.

We performed a 3-fold cross validation with the training set to accumulate misclassified examples for the human learning loop. Pat inspected these examples by using heat parse tree and encoded the declarative rules in \mathbf{W}_H (Table 3). The encoded declarative rules in \mathbf{W}_H are encoded from this example (Figure 1) and then injected as rows in matrix \mathbf{W}_H as described in Section 3.4

We compared three systems: *BoW* that contains only the word embedding used as a bag-of-words; *PureNN* that is the system without human knowledge; and *HumNN* that is the full system with Pat's declarative knowledge.

Table 3. Pat-in-the-Loop's f-measure

	f-measure	
	micro avg	macro avg
<i>BoW</i>	0.84	0.84
<i>PureNN</i>	0.93	0.91
<i>HumNN</i>	0.93	0.92

4.2. Results and Discussion

Results of our pilot experiment show important facts that we will examine in the following, focusing also on the limitations of this analysis.

Distributed tree encoders positively introduce syntactic information in neural networks: 0.84 to 0.93 of improvement in f-measure from *BoW* to *PureNN* (Table 3). This confirms a general trend observed in a similar experiment carried out in other classification tasks observed in [40].

The analysis of the errors in the training set produced very reasonable rules for two specific classes: Abbreviation (ABBR) and Numerical (NUM) (Table 4). For what concerns the abbreviation class, Pat selected very reasonable rules such as a question asks for the explanation of abbreviation if it contains parse subtrees representing the verbal phrases “stand for”, “mean” or the noun phrases containing the adjective “full” or the noun “abbreviation”. For what concerns the NUM class, rules are fairly more specific or definitely more general. Important indicators that a question is asking for a numerical answer are, respectively, that the question contains WH-noun-phrases “What debts” or contains noun phrases which are a sequence of two proper nouns, a possessive ending, and another noun, that is, $(NP (NP (NNP)(NNP)(POS))(NN))$. This latter is a very general rule. These rules are then used to build up the matrix W_H used in the model with human knowledge (*HumNN*).

Table 4. Rules manually extracted for the question classification dataset.

Class	Rule
ABBR	$(NP (NP (DT) (JJ \text{ full}) (NN)) (PP (IN)))$
ABBR	$(SQ (VBZ) (NP) (VP (VB \text{ stand}) (PP (IN \text{ for}))))$
ABBR	$(NN \text{ abbreviation})$
ABBR	$(VP (VB \text{ mean}))$
NUM	$(WHNP (WDT \text{ What}) (NNS \text{ debts}))$
NUM	$(NP (NP (NNP)(NNP)(POS))(NN))$

Pat could change positively the behavior of the system although global results of the model with human knowledge (*HumNN*) are similar and even slightly higher than those of *PureNN*. On the general results, the effect on the results of the system are small. In fact, the micro-average is 0.93 for both models is 0.93 and macro average is 0.92 for *HumNN* with respect to 0.91 of *PureNN*. Looking more specifically on the confusion matrix (Table 5 and 6), we may observe that Pat has changed the behavior of the system where he wanted. Since Pat aimed to manipulate the behavior of the system in favor of the classes ABBR and NUM, s/he focused the attention to examples where *PlainNN* fails. Pat's rules coded in W_H . After learning the new model *HumNN* disturbed by human declarative knowledge, results on the test set are encouraging. In fact, although the overall performance is unchanged, target classes have had positive improvement. Both ABBR and NUM have an additional positively classified example (Table 6). This tiny improvement suggests that the model can positively use declarative human knowledge. Finally, heat parse trees are informative. In fact, Pat could understand why some specific cases were misclassified and could select declarative rules to change the behavior of the system.

Table 5. PureNN's confusion matrices on Question Classification dataset (before human knowledge use).

	ABBR	ENTY	DESC	HUM	LOC	NUM
ABBR	6	0	3	0	0	0
ENTY	0	84	3	2	4	1
DESC	0	5	133	0	0	0
HUM	0	1	1	63	0	0
LOC	0	1	1	2	76	1
NUM	0	5	5	0	1	102

Table 6. HumNN's and confusion matrices on Question Classification dataset (after human knowledge use).

	ABBR	ENTY	DESC	HUM	LOC	NUM
ABBR	7	0	2	0	0	0
ENTY	0	83	5	3	2	1
DESC	0	3	135	0	0	0
HUM	0	3	0	62	0	0
LOC	0	4	1	1	74	1
NUM	0	3	4	1	2	103

Being a pilot study, the experiment has some intrinsic limitations. Clearly, the first limitation is the fact that the model has been experimented in a single and small dataset. However, this first pilot experiment is confirming our hypothesis. The second limitation is that we have not performed an ablation test on rules in Table 4. When adding external knowledge, introducing rules and consequently manipulating NNs processes could have negative impact on the system depending on the introduced rules to the system. However, in our pilot experiment, we introduced a very small set of rules which shows that Pat can obtain a positive variation of the behavior of the overall system. This is the major objective of the present study. In fact, globally, results of the pilot experiment confirmed our hypothesis: human can positively manipulate results of the system by inducing rules from the training set.

5. Conclusions and Future Work

In the line of understanding neural networks and trying to control their behavior besides using training examples, we presented Pat-in-the-Loop. Our model exploits syntactic information in neural networks by using *distributed tree encoders*, visualizes activation of syntactic information with *heat parse trees*, and encodes *declarative knowledge* in a neural network by keeping humans in the learning loop. Pat-in-the-Loop exploits Pat to understand why decisions are taken by a specific network and, hence, Pat can define specific rules to control the behavior of the neural network and s/he can understand why the decision has been made by looking at the activation of specific neurons for specific examples. According to our pilot study, Pat can obtain the desired change of the behavior of the overall Pat-in-the-Loop. Although giving encouraging results, our pilot experiment leaves some issues unanswered: the impact of the size of the dataset on the results and the impact of the quality of the introduced rules. These open issues will shape our future research. Hence, these encouraging results on a pilot study are a first “*declarative pat*” on neural networks applied to natural language processing, which may open a wide range of possible researches also, demonstrating as the humans in the loop is an important direction to ensure correctness, relevance, and cost-effective.

Our future plans stem on our recent result. We have expanded our approach with Kernel-inspired Encoder with Recursive Mechanism for Interpretable Trees (KERMIT) [40] and its visualizer

KERMITviz. Hence, our future goal is to analyze more carefully the interaction between the syntactic and semantic sources of information on heterogeneous tasks. Setting up a clear procedure for selecting positive declarative rules by means of ablation tests on a development set. The improvement given by this analysis may open the possibility of producing better rules for controlling the neural network. Then, we may better keep Human-in-the-loop of an Artificial Intelligence system [41].

Author Contributions: Software, D.O., P.T. and L.R.; Writing—original draft, D.O., L.R. and F.M.Z.; Writing—review & editing, F.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**, arXiv:1810.04805.
- Thompson, A. Google's Sentiment Analyzer Thinks Being Gay Is Bad. 2017. Available online: https://motherboard.vice.com/en_us/article/j5jmj8/google-artificial-intelligence-bias (accessed on 30 November 2020).
- Jessup, S.; Gibson, A.; Capiola, A.; Alarcon, G.; Borders, M. Investigating the Effect of Trust Manipulations on Affect over Time in Human-Human versus Human-Robot Interactions. 2020. Available online: https://www.researchgate.net/publication/339027805_Investigating_the_Effect_of_Trust_Manipulations_on_Affect_over_Time_in_Human-Human_versus_Human-Robot_Interactions (accessed on 30 November 2020). [CrossRef]
- Courtland, R. Bias detectives: The researchers striving to make algorithms fair. *Nature* **2018**, *558*, 357–360. [CrossRef]
- Zou, J.; Schiebinger, L. AI can be sexist and racist—It's time to make it fair. *Nature* **2018**, *559*, 324–326. [CrossRef]
- Kiritchenko, S.; Mohammad, S. Examining Gender and Race Bias in Two Hundred Sentiment Analysis Systems. In Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, *SEM@NAACL-HLT, New Orleans, LA, USA, 5–6 June 2018.
- Agrusti, G.; Damiani, V.; Pasquazi, D.; Carta, P. Reading mathematics at school. Inferential reasoning on the Pythagorean Theorem [Leggere la matematica a scuola. Percorsi inferenziali sul teorema di Pitagora]. *Cadmo* **2015**, *23*, 61–85. [CrossRef]
- Pasquazi, D. Capacità sensoriali e approccio intuitivo-geometrico nella preadolescenza: Un'indagine nelle scuole. *Cadmo* **2020**, *2020*, 79–96. [CrossRef]
- Dasgupta, S. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17*; Saul, L.K., Weiss, Y., Bottou, L., Eds.; MIT Press: Cambridge, MA, USA, 2005; pp. 337–344.
- Sener, O.; Savarese, S. Active Learning for Convolutional Neural Networks: A Core-Set Approach. *arXiv* **2018**, arXiv:1708.00489.
- Allen, G. Machine Learning: The View from Statistics. In Proceedings of the AAAS Annual Meeting, Houston, TX, 15 February 2019.
- Fink, M. Object Classification from a Single Example Utilizing Class Relevance Metrics. In *Advances in Neural Information Processing Systems 17*; Saul, L.K., Weiss, Y., Bottou, L., Eds.; MIT Press: Vancouver, CA, January 2005; pp. 449–456.
- Fei-Fei, L.; Fergus, R.; Perona, P. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 2006. [CrossRef]
- Wang, Y.; Yao, Q.; Kwok, J.; Ni, L.M. Generalizing from a Few Examples: A Survey on Few-Shot Learning. *arXiv* **2020**, arXiv:1904.05046.
- Jang, K.R.; Myaeng, S.H.; Kim, S.B. Interpretable Word Embedding Contextualization. Available online: <https://www.semanticscholar.org/paper/Interpretable-Word-Embedding-Contextualization-Jang-Myaeng/b8661fbfe31675f1fc90896458a796aca6c763c5> (accessed on 30 November 2020).

16. Jacovi, A.; Shalom, O.S.; Goldberg, Y. *Understanding Convolutional Neural Networks for Text Classification*. pp. 56–65. Available online: https://www.researchgate.net/publication/334115395_Understanding_Convolutional_Neural_Networks_for_Text_Classification (accessed on 30 November 2020). [CrossRef]
17. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. In *Computer Vision—ECCV 2014*; Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 818–833.
18. Li, J.; Chen, X.; Hovy, E.; Jurafsky, D. Visualizing and Understanding Neural Models in NLP. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016. [CrossRef]
19. Kahng, M.; Andrews, P.Y.; Kalro, A.; Chau, D.H. ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. *arXiv* **2017**, arXiv:1704.01942.
20. Ming, Y.; Cao, S.; Zhang, R.; Li, Z.; Chen, Y.; Song, Y.; Qu, H. Understanding Hidden Memories of Recurrent Neural Networks. In Proceedings of the 2017 IEEE Conference on Visual Analytics Science and Technology (VAST), Phoenix, AZ, USA, 3–6 October 2017.
21. Strobel, H.; Gehrmann, S.; Huber, B.; Pfister, H.; Rush, A.M. LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *arXiv* **2017**, arXiv:1606.07461.
22. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.
23. Vig, J. A multiscale visualization of attention in the transformer model. In Proceedings of the ACL 2019—57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August, 2019; pp. 37–42.
24. Wallace, E.; Tuyls, J.; Wang, J.; Subramanian, S.; Gardner, M.; Singh, S. AllenNLP Interpret: A Framework for Explaining Predictions of NLP Models. In Proceedings of the 2019 EMNLP, Hong Kong, China, 3–7 November 2019.
25. Hoover, B.; Strobel, H.; Gehrmann, S. exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformers Models. *arXiv* **2019**, arXiv:1910.05276.
26. Smilkov, D.; Thorat, N.; Nicholson, C.; Reif, E.; Viégas, F.B.; Wattenberg, M. Embedding projector: Interactive visualization and interpretation of embeddings. *arXiv* **2016**, arXiv:1611.05469.
27. 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.* **2017**, *13*. [CrossRef]
28. Collins, M.; Duffy, N. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, PA, USA, 6–12 July 2002; pp. 263–270.
29. Zanzotto, F.M.; Dell’Arciprete, L. Distributed Tree Kernels. In Proceedings of the 29th International Conference on Machine Learning, Edinburgh, UK, 26 June–1 July 2012.
30. Cortes, C.; Vapnik, V. Support Vector Networks. *Mach. Learn.* **1995**, *20*, 1–25. [CrossRef]
31. Cristianini, N.; Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*; Cambridge University Press: Cambridge, UK, 2000.
32. Johnson, W.; Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* **1984**, *26*, 189–206.
33. Plate, T.A. Holographic reduced representations. *IEEE Trans. Neural Netw.* **1995**, *6*, 623–641. [CrossRef]
34. 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* **2015**, *10*, 1–46. [CrossRef]
35. Li, X.; Roth, D. Learning Question Classifiers. Available online: <https://www.aclweb.org/anthology/C02-1150.pdf> (accessed on 30 November 2020).
36. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
37. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2017**, arXiv:1412.6980.
38. Keras Homepage. Available online: <https://keras.io> (accessed on 30 November 2020).
39. Klein, D.; Manning, C.D. Accurate Unlexicalized Parsing. Available online: <https://nlp.stanford.edu/~manning/papers/unlexicalized-parsing.pdf> (accessed on 30 November 2020).

40. Zanzotto, F.M.; Santilli, A.; Ranaldi, L.; Onorati, D.; Tommasino, P.; Fallucchi, F. KERMIT: Complementing Transformer Architectures with Encoders of Explicit Syntactic Interpretations. Available online: <https://www.aclweb.org/anthology/2020.emnlp-main.18.pdf> (accessed on 30 November 2020).
41. Zanzotto, F.M. Viewpoint: Human-in-the-loop Artificial Intelligence. *J. Artif. Intell. Res.* **2019**, *64*, 243–252. [CrossRef]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).