DIFFERENTIABLE RULE INDUCTION FROM RAW SEQUENCE INPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Rule learning-based models are widely used in highly interpretable scenarios for their transparent structures. Inductive logic programming (ILP) is a form of machine learning that induces rules from facts and keeps the interpretability. Differentiable ILP models enhance their learning ability in a robust and scalable manner with the advantages of neural networks. However, most differentiable ILP methods learn from symbolic datasets. Learning from raw data needs an ILP model to tackle the label leakage problem: The inability to map continuous inputs to symbolic variables without explicit supervision. In this work, we incorporate a self-supervised differentiable clustering model and a novel differentiable ILP model to learn from raw data without leaking the labels. The learned rules describe the raw data with its features. We demonstrate that our method learns generalized rules from time series and images intuitively and precisely.

1 INTRODUCTION

The deep learning models have obtained impressive performances on tabular classification, time series forecasting, image recognition, etc. While in highly trustworthy scenarios such as health care, finance, and policy-making process (Doshi-Velez & Kim, 2017), lacking explanations for decision-making prevents the applications of these complex deep learning models. However, the rule-learning models have interpretability intrinsically to explain the classification process. Inductive logic programming (ILP) is a form of logic-based machine learning that aims to learn logic programs for generalization and interpretability from training examples and background knowledge (Cropper et al., 2022). Traditional ILP methods design deterministic algorithms to induce rules from symbolic data to more generalized formal symbolic first-order languages (Quinlan, 1990; Blockeel & Raedt, 1998). However, these symbolic ILP methods face robustness and scalability problems when learning from large-scale and ambiguous datasets (Evans et al., 2021; Hocquette et al., 2024). With the sake of robustness of neural networks, the neuro-symbolic ILP by combining neural networks and ILP methods can learn from noisy data (Evans & Grefenstette, 2018; Manhaeve et al., 2018; Gao et al., 2022a) and can be applied to large-scale datasets (Yang et al., 2017; Gao et al., 2024; Phua & Inoue, 2024). However, existing neuro-symbolic ILP methods are mainly learned from discrete symbolic data or fuzzy symbolic data that the likelihoods are generated from a pre-trained neural network module (Evans et al., 2021; Shindo et al., 2023). Learning logic programs from raw data is prevented because of the label leakage problem, which is common in neuro-symbolic research (Topan et al., 2021): The leakage happens by introducing labels of ground objects for inducing rules (Evans & Grefenstette, 2018; Shindo et al., 2023). In fact, generating rules to describe objects in raw data without label information is necessary, especially when some objects are easily overlooked or lack labels yet are important for describing the data.

In this study, we propose *Neural Rule Learner* (NeurRL) that specifically focuses on learning logic programs directly from raw sequences, such as time series data and flattened image data. Compared to previous approaches with label leakage, which first obtain object labels from pre-trained neural networks and then use differentiable ILP methods to induce rules supervised by these labels, our method does not use a pre-trained neural network to generate symbolic labels. Instead, we use unsupervised machine learning methods, specifically differentiable clustering, to discretize the data into different features. Then, a differentiable rule-learning module is applied to find rules that describe input classes based on the distributions of the corresponding features. Thus, the model can be trained efficiently in a fully differentiable manner while avoiding the label leakage problem.

The detailed contributions of this study are as follows: Firstly, we formally defined the ILP learning task from raw inputs based on the learning from the interpretation transition setting of ILP. Secondly, we design a fully differentiable learning framework from raw sequences to symbolic rules, where we also propose a novel interpretable rule-learning module with multiple dense layers. Lastly, we test the model on various time series data and image data to prove the effectiveness and interpretability of the model.

2 RELATED WORK

Inductive logic programming (ILP), introduced by Muggleton & Feng (1990); Muggleton & Raedt (1994), learns logic programs from symbolic positive and negative examples with background knowledge. Inoue et al. (2014) proposed learning from interpretation transitions, and Phua & Inoue (2021) applied ILP to Boolean networks. Manhaeve et al. (2018); Evans & Grefenstette (2018) adapted neural networks for differentiable and robust logic program learning, while Gao et al. (2022b) introduced a neural network-based ILP model for learning from interpretations. This model was later extended for scalable learning from knowledge graphs (Gao et al., 2022a; 2024). Similarly, Liu et al. (2024) proposed a deep neural network for inducing mathematical functions. In our work, we present a novel neural network-based model for learning logic programs from raw numeric inputs.

In the raw input domain, Evans & Grefenstette (2018) proposed ∂ILP to learn rules from symbolic relational data, using a pre-trained neural network to map raw input data to symbolic labels. Unlike ∂ILP, which enforces a strong language bias by predefining logic templates and limiting the number of rule atoms, NeurRL uses only predicate types as language bias. Similarly, Evans et al. (2021) used pre-trained networks to map sensory data to disjunctive sequences, followed by binary neural networks to learn rules. Shindo et al. (2023) introduced αILP, leveraging object recognition models to convert images into symbolic atoms and employing top-k searches to pre-generate clauses, with neural networks optimizing clause weights. Our approach avoids pre-trained large-scale neural networks for mapping raw inputs to symbolic representations. Instead, we propose a fully differentiable framework to learn rules directly from raw numeric data. Additionally, unlike the memory-intensive rule candidate generation required by ∂ILP and αILP, NeurRL eliminates this step, enhancing scalability.

Adapting autoencoder and clustering methods in the neuro-symbolic domain shows promise. Sansone & Manhaeve (2023) applies conventional clustering on input embeddings for deductive logic programming tasks. Misino et al. (2022) and Zhan et al. (2022) use autoencoders and embeddings to calculate probabilities for predefined symbols to complete deductive logic programming and program synthesis tasks. In our approach, we use an autoencoder to learn representations for sub-areas of raw inputs, followed by a differentiable clustering method to assign ground atoms to similar patterns. The differentiable rule-learning module then searches for rule embeddings with these atoms in a bottom-up manner (Cropper & Dumancic, 2022). Similarly, DIFFNAPS, proposed by Barbiero et al. (2022), also uses an autoencoder to build hidden features and explain raw inputs. Additionally, BotCL, introduced by Wang et al. (2023), uses attention-based features to explain the ground truth class. However, the logical connections between the features used in DIFFNAPS and BotCL to describe the ground truth class are unclear. In contrast, rule-based explainable models like NeurRL use feature conjunctions to describe the ground truth class.

Azzolin et al. (2023) use a post-hoc rule-based explainable model to globally explain raw inputs based on local explanation results. In contrast, our model directly learns rules from raw inputs, and the performance of NeurRL is not influenced by other global explainable models. Das et al. (1998) used clustering to split sequence data into subsequences and symbolize each subsequence for rule discovery. Our model uses a fully differentiable framework that combines clustering and rule-learning methods to discover rules from sequence and image data, with the rule-learning module providing gradient information to prevent cluster collapse (Sansone, 2023). He et al. (2018) viewed similar subsequences, called motifs, as potential rule bodies. In our approach, we do not impose a limit on the number of body atoms in a rule. Wang et al. (2019) introduced SSSL, which learns rules from sequence data using shapelets as body atoms, maximizing dataset information gain. Our model expands on this by using raw data subsequences as rule body atoms and evaluating rule quality with precision and recall, a feature not present in SSSL.

3 PRELIMINARIES

3.1 LOGIC PROGRAMS AND INDUCTIVE LOGIC PROGRAMMING

A first-order language $\mathcal{L}$ is a tuple (D, F, C, V) (Lloyd, 1984), where D is a set of predicates, F is a set of function symbols, C is a set of constants, and V is a set of variables. A term is a constant, a variable, or an expression $f(t_1, t_2, \ldots, t_n)$ where f is a n-ary function symbol and $t_1, t_2, \ldots, t_n$ are terms. An atom is a formula $p(t_1, t_2, \ldots, t_n)$, where p is an n-ary predicate symbol and $t_1, t_2, \ldots, t_n$ are terms. A ground atom, or simply a fact, is an atom with no variables. A literal is an atom or its negation. A positive literal is just an atom, and a negative literal is the negation of an atom. A clause is a finite disjunction ($\vee$) of literals. A rule (or definite clause) is a clause with exactly one positive literal. If $\alpha_h, \alpha_1, \alpha_2, \ldots, \alpha_n$ are atoms, then $\alpha_h \vee \neg\alpha_1 \vee \neg\alpha_2 \vee \cdots \vee \neg\alpha_n$ is a rule. We rewrite each rule r in the following form:

$$\alpha_h \leftarrow \alpha_1, \alpha_2, \ldots, \alpha_n, \tag{1}$$

where the atom α_h is called the head denoted as head(r), the set of atoms $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is called the body denoted as body(r), and each atom in a body of a rule is called a body atom. A logic program P is a set of rules. In first-order logic, a substitution is a finite set of the form $\{v_1/t1, v_2/t_2, \ldots, v_n/t_n\}$, where each v_i is variable, each t_i is a term distinct from v_i and the variables $v_1, v_2, \ldots, v_n$ are distinct (Lloyd, 1984). A substitution is called a ground substitution if the t_i are all ground terms. In addition, the set of ground instances of all rules in a logic program P is denoted as ground(P).

The Herbrand base B_P for a logic program P is the set of all ground atoms whose predicate symbols occur in P, and an interpretation I is a subset of B_P with the true ground atoms (Lloyd, 1984). Given an interpretation I, the immediate consequence operator $T_P \colon 2^{B_P} \to 2^{B_P}$ of a definite logic program P is defined as: $T_P(I) = \{\text{head}(r) \mid r \in \text{ground}(P), \text{body}(r) \subseteq I\}$ (Apt et al., 1988). In addition, a logic program P with m rules sharing the same head atom α_h, indexed by h, and n possible body atoms across all rules can be represented as a logic program matrix $\mathbf{M}_P \in [0, 1]^{m \times n}$. Each element a_{kj} in $\mathbf{M}_P$ is defined as follows (Gao et al., 2022a): If the k-th rule is $\alpha_h \leftarrow \alpha_{j_1} \wedge \alpha_{j_2} \wedge \cdots \wedge \alpha_{j_p}$, then $a_{kj_i} = l_i$, where $l_i \in (0, 1)$ and $\sum_{s=1}^{p} l_s = 1$ ($1 \leq i \leq p,\ 1 < p,\ 1 \leq j_i \leq n,\ 1 \leq k \leq m$). If the k-th rule is $\alpha_h \leftarrow \alpha_j$, then $a_{kj} = 1$. Otherwise, $a_{kj} = 0$. Each row in $\mathbf{M}_P$ represents a rule in a logic program P, and each column represents a body atom in P. An interpretation vector $\mathbf{v}_I \in \{0, 1\}^n$ corresponds to a substitution or an interpretation I, where if the ground atom grounded from i-th atom is true in the interpretation I, then $\mathbf{v}_I[i] = 1$; otherwise, $\mathbf{v}_I[i] = 0$. Then, given an interpretation vector $\mathbf{v}_I$, an algebraic immediate consequence operator $D_P : \{0, 1\}^n \to \{0, 1\}^m$ is defined as follows (Sakama et al., 2021):

$$D_P(\mathbf{v}_I) = \theta(\mathbf{M}_P \mathbf{v}_I), \tag{2}$$

where the function θ is a threshold function: $\theta(x) = 1$ if $x \geq 1$, otherwise $\theta(x) = 0$. Consequently, the Boolean value of the head atom $v(\alpha_h)$ is calculated by $v(\alpha_h) = \bigvee_{i=1}^{m} D_P(\mathbf{v}_I)[i]$. Furthermore, Gao et al. (2024) uses a differentiable threshold function to replace the threshold function $\theta(x)$ and the fuzzy disjunction $\tilde{\bigvee}_{i=1}^{m} \mathbf{x}[i] = 1 - \prod_{i=1}^{m} \mathbf{x}[i]$ to replace the logical disjunction operator to obtain the differentiable immediate consequence operator.

Inductive logic programming aims to induce logic programs from training examples and background knowledge (Muggleton et al., 2012). The learning settings of ILP include learning from entailments (Evans & Grefenstette, 2018), interpretations (Raedt & Dehaspe, 1997), proofs (Passerini et al., 2006), and interpretation transitions (Inoue et al., 2014). In the paper, we use the learning from interpretation transitions setting: Given a set $E \subseteq 2^{B_P} \times 2^{B_P}$ consisting of pairs of interpretations (I, J), the goal is to learn a logic program P such that $T_P(I) = J$ holds for all $(I, J) \in E$.

3.2 SEQUENCE DATA AND DIFFERENTIABLE CLUSTERING METHOD

A raw input can be described as follows: Given an instance $(\mathbf{x}, \mathbf{y})$, $\mathbf{x} \in \mathbb{R}^{T_1 \times T_2 \times \cdots \times T_d}$ is real-valued observations representing the input, d is the number of dimensions, and the class $\mathbf{y} \in \{0, 1\}^C$, with C being the predefined number of classes. A sequence data is a type of raw input where the input, denoted as $\mathbf{x}$, consists of ordered real-valued observations in one dimension. Hence, each sequence input $\mathbf{x} \in \mathbb{R}^T$, where T is the number of points in the sequence data (Wang et al., 2019).

A subsequence $\mathbf{s}_i$ with the length l on sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ is a contiguous sequence $(x_i, \ldots, x_{i+l-1})$ (Das et al., 1998). Hence, all possible subsequence with length l include $\mathbf{s}_1$, …, $\mathbf{s}_{T-l+1}$.

The clustering method can be used to discover a new set of categories (Rokach & Maimon, 2005). In the paper, we adapted the differentiable k-means clustering method (Fard et al., 2020) to group raw data $\mathbf{x} \in \mathbf{X}$. Firstly, to concentrate high-level features in embedding space, an autoencoder A is used to generate the embeddings $\mathbf{h}_\gamma$ for input data $\mathbf{x}$, where the parameters in the autoencoder are represented by γ. Secondly, to discretize all data $\mathbf{X}$ into several clusters, K is set as the number of clusters to be obtained. Then, $\mathbf{r}_k \in \mathbb{R}^p$ indicates the representation of k-th cluster where p indicates the dimension of the representation, and $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_K\}$ is the set of all K representations of clusters. For any vector $\mathbf{y} \in \mathbb{R}^p$, the function $c_f(\mathbf{y}; \mathcal{R})$ returns the closest representation of the vector $\mathbf{y}$ based on a fully differentiable distance function f. Then, the differentiable k-means problems are defined as follows:

$$\min_{\mathcal{R},\gamma} \sum_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x}, A(\mathbf{x};\gamma)) + \lambda \sum_{k=1}^{K} f(\mathbf{h}_\gamma(\mathbf{x}), \mathbf{r}_k) G_{k,f}(\mathbf{h}_\gamma(\mathbf{x}), \alpha; \mathcal{R}), \tag{3}$$

where the parameter λ regulates the trade-off between seeking good representations for $\mathbf{x}$ and the representations that are useful for clustering proposes. The weight function G is a differentiable minimum function proposed by Jang et al. (2017):

$$G_{k,f}(\mathbf{h}_\gamma(\mathbf{x}), \alpha; \mathcal{R}) = \frac{e^{-\alpha f(\mathbf{h}_\gamma(\mathbf{x}), \mathbf{r}_k)}}{\sum_{k'=1}^{K} e^{-\alpha f(\mathbf{h}_\gamma(\mathbf{x}), \mathbf{r}_{k'})}}, \tag{4}$$

where $\alpha \in [0, +\infty)$. The larger α makes the different minimum function like the discrete minimum function. However, the smaller α makes the training process smoother.

4 METHODS

4.1 PROBLEM STATEMENT AND FORMALIZATION

In this subsection, we formally define the learning problem from raw inputs with the learning from the interpretation transition setting of ILP and define the language bias of rules to describe sequence data. We use a neuro-symbolic description method proposed by Marconato et al. (2023) to describe a raw input: (i) assume that the label $\mathbf{y}$ of a raw input depend entirely on the state of K symbolic concepts $B = (a_1, a_2, \ldots, a_K)$ capturing high-level aspects of the input $\mathbf{x}$. (ii) The concepts B depend on the sub-symbolic input $\mathbf{x}'$ in an intricate manner and are best extracted using deep learning. (iii) The way in which the labels $\mathbf{y}$ depend on the concepts B can also be specified by prior knowledge $\mathcal{B}$, necessitating reasoning during inference.

In the paper, we treat each subset of raw input as a constant and treat each concept as a ground atom. For instance, in a sequence data $\mathbf{x} \in \mathbf{X}$, if there is a concept where the value increases from time point 0 to time point 10, the corresponding ground atom is $\texttt{increase}(\mathbf{x}\,[0\colon 10])$. We define *body symbolization function* L_b that transfers a continuous sequence $\mathbf{x}$ to discrete symbolic concepts in $\mathbf{x}$. Then, all symbolic concepts B in all raw inputs $\mathbf{X}$ are $B = L_b(\mathbf{X})$. Besides, we set another function called *head symbolization function* L_h to map an input $\mathbf{x}$ to a set of atoms. When the class of $\mathbf{x}$ is target class denoted as t, then $L_h(\mathbf{x}) = \{h_t\}$; otherwise, $L_h(\mathbf{x}) = \emptyset$. In this paper, we use the target atom, denoted as h_t, to represent the target class of an instance. Specifically, when h_t is true, the instance belongs to the positive class; otherwise, it belongs to the negative class. Consequently, a logic program P describing a binary class raw inputs[1] consists of the rules with the same head atom h_t. Then, we regard the set with all symbolic concepts B in all raw inputs $\mathbf{X}$ and a head atom h_t as the Herbrand base B_P in a logic program P. Hence, for a raw input $\mathbf{x}$, $L_b(\mathbf{x})$ represents a set of symbolic concepts $I \subseteq B_P$ appearing in $\mathbf{x}$, which I can be regarded as an interpretation. To induce a logic program P from raw inputs, we define the learning task based on the learning from interpretation transition setting of ILP as follows:

[1]We can transfer multiple class raw inputs to the binary class by setting the class of interest as the target class and the other classes as the negative class.

Definition 1 (Learning from Raw Input) *Given a set of raw inputs* $\mathbf{X}$*, the learning task is to learn a logic program* P*, where* $T_P(L_b(\mathbf{x})) = L_h(\mathbf{x})$ *holds for all raw inputs* $\mathbf{x} \in \mathbf{X}$.

In the paper, we aim to learn rules to describe the target class with the body consisting of multiple sequence features. Each feature corresponds to a subsequence of the sequence data. Besides, each feature includes the pattern information and region information of the subsequence. Based on the pattern information, we further infer the mean value and tendency information of the subsequence. Specifically, we use the following rules to describe the sequence data with the target class:

$$h_t \leftarrow pattern_{i_1}(X_{j_1}), region_{k_1}(X_{j_1}), \ldots, pattern_{i_{n_1}}(X_{j_{n_2}}), region_{k_{n_3}}(X_{j_{n_2}}), \tag{5}$$

where the predicate $\texttt{pattern}_i$ indicates the i-th pattern in all finite patterns within all sequence data, the predicate $\texttt{region}_k$ indicates the k-th region in all regions in a sequence, and the variable X_j can be substituted by a subsequence of the sequence data. For example, $\texttt{pattern}_1(\mathbf{x}[0:5])$ and $\texttt{region}_0(\mathbf{x}[0:5])$ indicate that the subsequence $\mathbf{x}[0:5]$ matches the pattern with index one and belongs to the region with index zero, respectively. A pair of atoms, $\texttt{pattern}_i(X) \wedge \texttt{region}_k(X)$, corresponds to a feature within the sequence data. In this pair, the variables are identical, with one predicate representing a pattern and the other representing a region. We infer the following information from the rules in format (5): In a sequence input $\mathbf{x}$, if all pairs of ground patterns and regions atoms substituted by subsequences in $\mathbf{x}$ are true, then the sequence input $\mathbf{x}$ belongs to the target class represented by the head atom h_t.

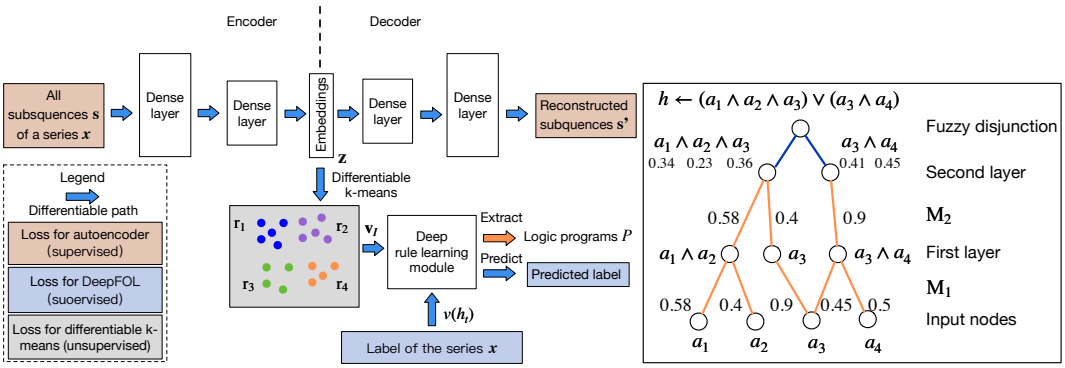
4.2 Differentiable symbolization process

In this subsection, to learn a logic program P describing the target class with pattern and region information with rules defined in the format (5) in a fully differentiable learning pipeline, we design a differentiable body symbolization function $\tilde{L}_b$ inspired by differentiable k-means (Fard et al., 2020) to transfer the numeric sequence data $\mathbf{x}$ to a *fuzzy interpretation vector* $\mathbf{v}_I \in [0,1]^n$. The fuzzy interpretation vector encodes the fuzzy values of all the ground patterns and region atoms, which are substituted by subsequences of inputs. In a fuzzy interpretation vector $\mathbf{v}_I$, if the i-th element has a higher value, then the i-th atom in the interpretation I tends to be true. Based on the head symbolization function $J = L_h(\mathbf{x})$, we can determine the Boolean value of target atom $v(h_t)$, which $v(h_t) = 1$ if $J = \{h_t\}$ and $v(h_t) = 0$ if $J = \emptyset$. Inspired by the learning from interpretation vector transitions (Gao et al., 2024), we can learn a logic program matrix $\mathbf{M}_P \in [0,1]^{m \times n}$ that $\tilde{\bigvee}_{i=1}^{m} D_P(\mathbf{v}_I)[i] = v(h_t)$ holds. Then, the rules in the format (5) can be extracted from $\mathbf{M}_P$, and these rules are generalized from the most specific clause, whose body atoms consist of all pattern and region predicates with finite variables.

The architecture of NeurRL is presented in Fig. 1a: To learn logic program P from the sequence data, we first divide each sequence input $\mathbf{x}$ into shorter subsequences $\mathbf{s}$ of length l, using a unit step stride. The encoder recognizes subsequence data $\mathbf{s}$ to an embedding space $\mathbf{z}$[2], and the decoder reconstructs the subsequence data $\mathbf{s}'$ from the embedding space $\mathbf{z}$. The differentiable k-means described in Section 3.2 distribute the embeddings $\mathbf{z}$ and consequently assign input subsequence $\mathbf{s}$ into different groups $\mathbf{r}$, and each group of subsequences having a similar pattern. Then, we can obtain the fuzzy interpretation vector $\mathbf{v}_I$ and Boolean values of target atom $v(h_t)$ corresponding to each sequence instance. Lastly, the differentiable rule-learning module can be applied to learn high-level rules describing the target class with $\mathbf{v}_I$'s as inputs and $v(h_t)$'s as labels corresponding to all sequence inputs.

Now, we describe the method to build the differentiable body symbolization function $\tilde{L}_b$ from sequence $\mathbf{x}$ to interpretation vector $\mathbf{v}_I$ as follows: Let K be the maximum number of clusters based on the differentiable k-means algorithm, each subsequence $\mathbf{s}$ with the length l in $\mathbf{x}$ and the corresponding embedding $\mathbf{z}$ has a cluster index c ($1 \leq c \leq K$), and each sequence input $\mathbf{x}$ can be transferred into a vector of cluster indexes $\mathbf{c} \in \{0,1\}^{K \times (|\mathbf{x}|-l+1)}$. Additionally, to incorporate temporal or spatial information into the predicates of the target rules, we divide the entire sequence data into P equal regions. The region of a subsequence $\mathbf{s}$ is determined by the location of its first point, $\mathbf{s}[0]$. For each subsequence $\mathbf{s}$, we calculate its *cluster index vector* $\mathbf{c}_s \in [0,1]^K$ using the weight function

[2]Note that based on different types of input data, the structure of the encoder can be different. We use fully connected layers as the encoder and decoder in the paper.



(a) Learning pipeline (b) Deep rule-learning module

Figure 1: The learning pipeline of NeurRL and the rule-learning module

$G_{k,f}$ as defined Eq. (3), where $\mathbf{c}_s[k] = G_{k,f}(\mathbf{h}_\gamma(\mathbf{s}), \alpha; \mathcal{R})$ $(1 \leq k \leq K$ and $\sum_{i=1}^{K} \mathbf{c}_s[i] = 1)$. A higher value in the i-th element of $\mathbf{c}_s$ indicates that the subsequence $\mathbf{s}$ is more likely to be grouped into the i-th cluster. Hence, we can transfer the sequence input $\mathbf{x} \in \mathbb{R}^T$ to *cluster index tensor* with the possibilities of cluster indexes of subsequences in all regions $\mathbf{c}_x \in [0,1]^{K \times l_p \times P}$, where l_p is the number of subsequence in one region of input sequence data $\lceil |\mathbf{x}| / P \rceil$. To calculate the cluster index possibility of each region, we sum the likelihood of cluster index of all subsequence within one region in cluster index tensor $\mathbf{c}_x$ and apply softmax function to build *region cluster matrix* $\mathbf{c}_p \in \mathbb{R}^{K \times P}$ as follows:

$$\mathbf{c}[i,j] = \sum_{k=1}^{l_p} \mathbf{c}_x[i,k,j], \ \mathbf{c}_p[i,j] = \frac{e^{\mathbf{c}[i,j]}}{\sum_{i=1}^{K} e^{\mathbf{c}[i,j]}}. \tag{6}$$

Since the region cluster matrix $\mathbf{c}_p$ contains clusters index for each region and each cluster corresponds to a pattern, we flatten $\mathbf{c}_p$ into a fuzzy interpretation vector $\mathbf{v}_I$, which serves as the input to the deep rule-learning module of NeurRL.

4.3 Differentiable Rule-Learning Module

In this section, we define the novel deep neural network-based rule-learning module to induce rules from fuzzy interpretation vectors. Based on the label of the sequence input $\mathbf{x}$, we can determine the Boolean value of target atom $v(h_t)$. Then, using fuzzy interpretation vectors $\mathbf{v}_I$ as inputs and Boolean values of the head atom $v(h_t)$ as labels y, we build a novel neural network-based rule-learning module as follows: Firstly, each input node receives the fuzzy value of a pattern or region atom stored in $\mathbf{v}_I$. Secondly, one output node in the final layer reflects the fuzzy values of the head atom. Then, the neural network consists of k dense layers and one disjunction layer. Lastly, let the number of nodes in the k-th dense layer be m, then the forward computation process is formulated as follows:

$$\hat{y} = \tilde{\bigvee}_{i=1}^{m} \left(g_k \circ g_{k-1} \circ \cdots \circ g_1(\mathbf{v}_I)\right)[i], \tag{7}$$

where the i-th dense layer is defined as:

$$g_i(\mathbf{x}_{i-1}) = \frac{1}{1-d} \operatorname{ReLU}\left(\mathbf{M}_i \mathbf{x}_{i-1} - d\right), \tag{8}$$

with d as the fixed bias[3]. The matrix $\mathbf{M}_i$ is the softmax-activated of trainable weights $\tilde{\mathbf{M}}_i \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ in each dense layer of the rule-learning module:

$$\mathbf{M}_i[i,j] = \frac{e^{\tilde{\mathbf{M}}_i[i,j]}}{\sum_{j=1}^{n_{\text{in}}} e^{\tilde{\mathbf{M}}_i[i,j]}}. \tag{9}$$

[3]In our experiments, we set the fixed bias as 0.5.

With the differentiable body symbolization function $\tilde{L}_b$ and the rule-learning module of NeurRL denoted as N_R, we now define a target function that integrates the autoencoder module, clustering module, and rule-learning module as follows:

$$\min_{\mathcal{R},\gamma_e,\gamma_l} \sum_{\mathbf{s}\in\mathbf{x},\mathbf{x}\in\mathbf{X}} f_1\left(\mathbf{s}, A\left(\mathbf{s};\gamma_e\right)\right) + \lambda_1 \sum_{k=1}^{K} f_1\left(\mathbf{h}_{\gamma_e}(\mathbf{s}), \mathbf{r}_k\right) G_{k,f_1}(\mathbf{h}_{\gamma_e}(\mathbf{s}), \alpha; \mathcal{R}) + \lambda_2 f_2\left(N_R\left(\tilde{L}_b\left(\mathbf{x}\right);\gamma_l\right), y\right), \tag{10}$$

where γ_e and γ_l represent the trainable parameters in the encoder and rule-learning module, respectively. The loss function f_1 and f_2 are set to mean square error loss and cross-entropy loss correspondingly. The parameters λ_1 and λ_2 regulate the trade-off between finding the concentrated representations of subsequences, the representations of clusters for obtaining precise patterns, and the representations of rules to generalize the data[4]. Fig. 1a illustrates the loss functions defined in the target function (10). The supervised loss functions are applied to the autoencoder (highlighted in orange boxes) and the rule-learning module (highlighted in blue boxes), respectively. The unsupervised loss function is applied to the differentiable k-means method.

To train the model, we first pre-train an autoencoder to obtain subsequence embeddings, then initialize the cluster embeddings using k-means clustering (Lloyd, 1982) based on these embeddings. Finally, we jointly train the autoencoder, clustering model, and rule-learning model using the target function (10) to optimize the embeddings, clusters, and rules simultaneously.

We analyze the interpretability of the rule-learning module N_R. In the logic program matrix $\mathbf{M}_P$ defined in Section 3.1, the sum of non-zero elements in each row, $\sum_{j=1}^{n} \mathbf{M}_P[i,j]$, is normalized to one, matching the threshold in the function θ in Eq. (2). The conjunction of the atoms corresponding to non-zero elements in each row of $\mathbf{M}_P$ can serve as the body of a rule. Similarly, in the rule-learning module of NeurRL, the sum of the softmax-activated weights in each layer is also one. Due to the properties of the activation function $\text{ReLU}(x-d)/(1-d)$ in each node, a node activates only when the sum of the weights approaches one; otherwise, it deactivates. This behavior mimics the threshold function θ in Eq. (2).

Similar to the logic program matrix $\mathbf{M}_P$, the softmax-activated weights in each layer also have interpretability. When the fuzzy interpretation vector and Boolean value of the target atom fit the forward process in Eq. (7), the atoms corresponding to non-zero elements in each row of the i-th dense softmax-activated weight matrix $\mathbf{M}_i$ form a conjunction. From a neural network perspective, the j-th node in the i-th dense layer, denoted as $n_j^{(i)}$, represents a conjunction of atoms from the previous $(i-1)$-th dense layer (input layer). The likelihood of these atoms appearing in the conjunction is determined by the softmax-activated weights $\mathbf{M}_i[j,:]$ connecting to node $n_j^{(i)}$. In the final k-th dense layer, the disjunction layer computes the probability of the target atom h_t. The higher likelihood conjunctions, represented by the nodes in the k-th layer, form the body of the rule headed by h_t. To interpret the rules headed by h_t, we compute the product of all softmax-activated weights $\mathbf{M}_i$ as the *program tensor*: $\mathbf{M}_P = \prod_{i=1}^{k} \mathbf{M}_i$, where $\mathbf{M}_P \in [0,1]^{m\times n}$. The program tensor has the same interpretability as the program matrix, with high-value atoms in each row forming the rule body and the target atom as the rule head. The number of nodes in the last dense layer m determines the number of learned rules in one learning epoch. Fig. 1b shows a neural network with two dense layers and one disjunction layer in blue. The weights in orange represent significant softmax-activated values, with input nodes as atoms and hidden nodes as conjunctions. Multiplying the softmax weights identifies the atoms forming the body of a rule headed by the target atom.

We use the following method to extract rules from program tensor $\mathbf{M}_P$: We set multiple thresholds $\tau \in [0,1]$ when the value of the element in each row of $\mathbf{M}_P$ is larger than the threshold τ, then we regard the atom corresponding to these elements as one of body atom in the rule with the format (5). We calculate the precision and recall of rules on the discretized fuzzy interpretation vector generated from the test dataset, and the threshold to discrete continuous fuzzy interpretation vector $\mathbf{v}_I$ is set to 0.5. Then, we keep the high-precision rules as the output. The precision and recall of a rule are defined as follows:

$$\text{precision} = \frac{n_{\text{body} \wedge \text{head}}}{n_{\text{body}}},\ \text{recall} = \frac{n_{\text{body} \wedge \text{head}}}{n_{\text{head}}}, \tag{11}$$

[4]In our experiments, we assigned equal weights to finding representations, identifying clusters, and discovering rules by setting $\lambda_1 = \lambda_2 = 1$.

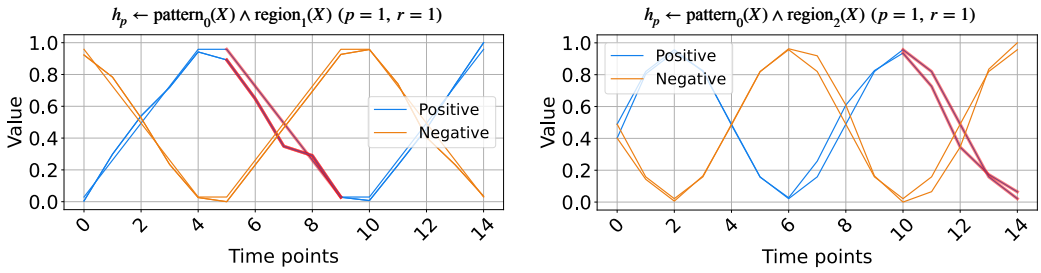
where $n_{\text{body} \wedge \text{head}}$ denotes the number of discretized fuzzy interpretation vectors $\mathbf{v}_I$ that satisfy the rule body and have the target class as the label. Similarly, n_{body} represents the number of discretized fuzzy interpretation vectors that satisfy the rule body, while n_{head} refers to the number of instances with the target class. When obtaining rules in the format (5), we can highlight the subsequences satisfying the pattern and region predicates above on the raw inputs for more intuitive interpretability.

5 Experimental Results

5.1 Learning from Synthetic Data

In this subsection, we evaluate the model on synthetic time series data based on triangular pulse signals and trigonometric signals. Each signal contains two key patterns, increasing and decreasing, with each pattern having a length of five units. To test NeurRL's learning capability on a smaller dataset, we set the number of inputs in both the positive and negative classes to two for both the training and test datasets. In each class, the difference between two inputs at each time point is a random number drawn from a normal distribution with a mean of zero and a variance of 0.1. The positive test inputs are plotted in blue, and the negative test inputs in orange in Fig. 2. We set both the length of each region and the subsequence length to five units, and the number of clusters is set to three in this experiment. NeurRL is tasked with learning rules to describe the positive class, represented by the head atom h_p. If the ground body atoms, substituted by subsequences of an input, hold true, the head atom h_p is also true, indicating that the target class of the input is positive.

The rules with 1.0 precision (p) and 1.0 recall (r) are shown in Fig. 2. The rule in Fig. 2a states that when $\texttt{pattern}_0$ (cluster index 0) appears in $\texttt{region}_1$ (from time points 5 to 9), the time series label is positive. Similarly, the rule in Fig. 2b indicates that when $\texttt{pattern}_0$ appears in $\texttt{region}_2$ (from time points 10 to 14), the label is positive. We highlight subsequences that satisfy the rule body in red in Fig. 2, inferring that the pattern's trend is decreasing. These red patterns perfectly distinguish positive from negative inputs.



(a) The signals based on triangular pulse

(b) The signals based on trigonometry function

Figure 2: The synthetic data and the learned rules

5.2 Learning from UCR Datasets

In this subsection, we experimentally demonstrate the effectiveness of NeurRL on 13 randomly selected datasets from UCR (Dau et al., 2019), as used by Wang et al. (2019). To evaluate NeurRL's performance, we consider the classification accuracy from the rules extracted by the deep rule-learning module (denoted as NeurRL(R)) and the classification accuracy from the module itself (denoted as NeurRL(N)). The number of clusters in this experiment is set to five. The subsequence length and the number of regions vary for each task. The subsequence length corresponds to the potential pattern length in the raw time series, while the period length relates to the overall length of the time series. We set the number of periods to approximately 10 for time series data. Additionally, the subsequence length is set to range from two to five, depending on the specific subtask.

The baseline models include SSSL (Wang et al., 2019), Xu (Xu & Funaya, 2015), and BoW (Wang et al., 2013). SSSL uses regularized least squares, shapelet regularization, spectral analysis, and pseudo-labeling to auto-learn discriminative shapelets from time series data. Xu's method constructs a graph to derive underlying structures of time series data in a semi-supervised way. BoW generates

a bag-of-words representation for time series and uses SVM for classification. Statistical details, such as the number of classes (C.), inputs (I.), series length, and comparison results are shown in Tab. 1, with the best results in bold and second-best underlined. The NeurRL(N) achieves the most best results, with seven, and NeurRL(R) achieves five second-best results.

To demonstrate the benefits of the fully differentiable learning pipeline from raw sequence inputs to symbolic rules, we compare the accuracy of rules and running times (in seconds) between NeurRL and its deep rule-learning module using the non-differentiable k-means clustering algorithm (Lloyd, 1982). We use the same parameters to split the time series into subsequences. The non-differentiable k-means generates clusters based on raw subsequence data. Results in Tab. 2 show that the fully differentiable pipeline, based on neural networks, significantly reduces running time without sacrificing rule accuracy in most cases.

Table 1: Classification accuracy on 13 binary UCR datasets with different models

Dataset	C.	I.	Length	Xu	BoW	SSSL	NeurRL(R)	NeurRL(N)
Coffee	2	56	286	0.588	0.620	0.792	<u>0.964</u>	**1.000**
ECG	2	200	96	0.819	**0.955**	0.793	0.820	<u>0.850</u>
Gun point	2	200	150	0.729	**0.925**	0.824	0.740	<u>0.873</u>
ItalyPow.Dem.	2	1096	24	0.772	0.813	**0.941**	<u>0.926</u>	0.879
Lighting2	2	121	637	0.698	0.721	**0.813**	0.689	<u>0.738</u>
CBF	3	930	128	0.921	0.873	**1.000**	0.909	<u>0.930</u>
Face four	4	112	350	0.833	0.744	0.851	<u>0.914</u>	**0.964**
Lighting7	7	143	319	0.511	0.677	<u>0.796</u>	0.737	**0.878**
OSU leaf	6	442	427	0.642	0.685	0.835	<u>0.844</u>	**0.849**
Trace	4	200	275	0.788	**1.00**	**1.00**	0.833	<u>0.905</u>
WordsSyn	25	905	270	0.639	0.795	0.875	<u>0.932</u>	**0.946**
OliverOil	4	60	570	0.639	0.766	<u>0.776</u>	0.768	**0.866**
StarLightCurves	3	9236	2014	0.755	0.851	<u>0.872</u>	0.869	**0.907**
Mean accuracy				0.718	0.801	<u>0.859</u>	0.842	**0.891**

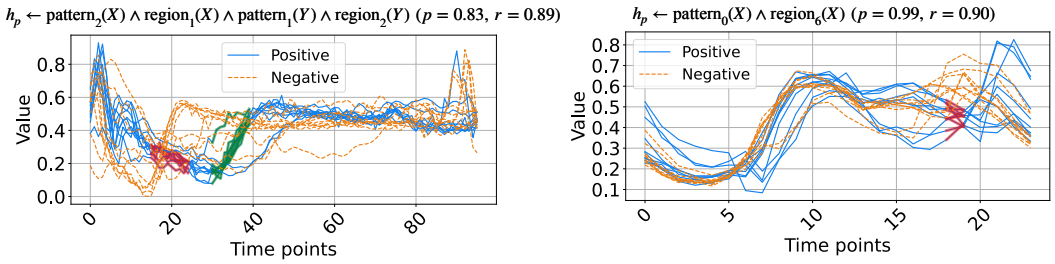
Table 2: Comparisons with non-differentiable k-means clustering algorithm

Dataset	Non-differentiable k-means		Differentiable k-means	
	accuracy	running time	accuracy	running time
Coffee	0.893	313	**0.964**	**42**
ECG	0.810	224	**0.820**	**65**
Gun point	**0.807**	102	0.740	**35**
ItalyPow.Dem.	0.845	114	**0.926**	**63**
Lighting2	0.672	1166	**0.689**	**120**

The learned rules from the ECG and ItalyPow.Dem. datasets in the UCR archive are shown in Fig. 3. In Fig. 3a, red highlights subsequences with the shape $\texttt{pattern}_2$ in the region $\texttt{region}_1$, while green highlights subsequences with the shape $\texttt{pattern}_1$ in the region $\texttt{region}_2$. The rule suggests that when data decreases between time points 15 to 25 and then increases between time points 30 to 40, the input likely belongs to the positive class. The precision and recall for this rule are 0.83 and 0.89, respectively. Similarly, in Fig. 3b, red highlights subsequences with the shape $\texttt{pattern}_0$ in the region $\texttt{region}_6$. The rule indicates that a lower value around time points 18 to 19 suggests the input belongs to the positive class, with precision and recall of 0.99 and 0.90, respectively.

5.3 Learning from images

In this subsection, we ask the model to learn rules to describe and discriminate two classes of images from MNIST datasets. We divide the MNIST dataset into five independent datasets, where each dataset contains the positive class presented in the upper row of Fig. 4 and the negative class



(a) A rule from ECG dataset

(b) A rule from ItalyPow.Dem. dataset

Figure 3: Selected rules from two UCR datasets

presented in the lower row of Fig. 4. For two-dimensional image data, we first flatten the image data to one-dimensional sequence data. Then, the sequence data can be the input for the NeurRL model to learn the rules. The lengths of subsequence and region are both set to three. Besides, the number of clusters is set to five. After generating the highlighted patterns based on the rules, we recover the sequence data to the image for interpreting these learned rules. We present the rule with the precision larger than 0.9 and the highlight features (or areas) corresponding to the learned rule in Fig. 4. Each highlight feature corresponds to a pair of region and pattern atoms in a learned rule. To interpret the rules in Fig. 4, we can interpret them like importance attention (Zhang et al., 2019), where the colored areas include highly discriminative information for describing positive inputs compared with negative inputs. For example, in the first column of Fig. 4, if the highlighted areas are in black at the same time, then the image class is one. Otherwise, the image class is zero. Compared with attention, we calculate the precision and recall to evaluate these highlight features quantitatively.

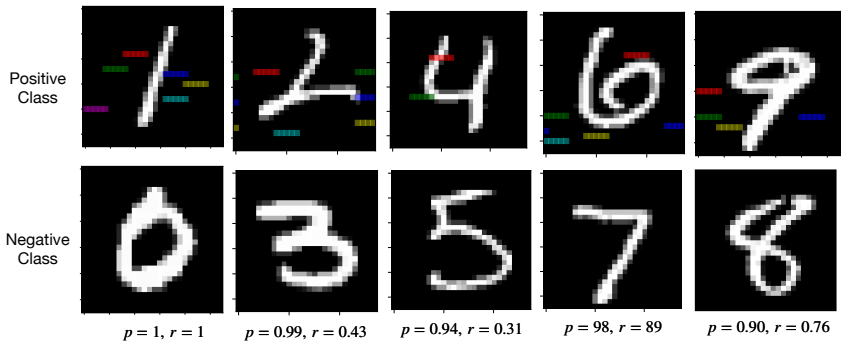


Figure 4: Rules from MNIST datasets

6 Conclusion

Inductive logic programming (ILP) is a rule-based machine learning method that supports data interpretability. Differentiable ILP offers advantages in scalability and robustness. However, label leakage remains a challenge when learning rules from raw data, as neuro-symbolic models require intermediate feature labels as input. In this paper, we propose a novel fully differentiable ILP model, Neural Rule Learner (NeurRL), which learns symbolic rules from raw sequences using a differentiable k-means clustering module and a deep neural network-based rule-learning module. The differentiable k-means clustering algorithm groups subcomponents of inputs based on the similarity of their embeddings, and the learned clusters can be used as input for the rule-learning module to induce rules that describe the ground truth class of input based on its features. The proposed model can not only learn rules from time series but also learn from images. Compared to other rule-based models, NeurRL achieves comparable classification accuracy while offering interpretability through quantitative metrics. Currently, the atoms in the learned rules describe subareas of a raw input. In the future, we aim to learn rules where an atom represents a set of instances, enabling the application of learning rules to explain digital semantics (Evans et al., 2021) without label leakage.

REFERENCES

Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pp. 89–148. Morgan Kaufmann, 1988.

Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global explainability of gnns via logic combination of learned concepts. In *Proceedings of the 11th International Conference on Learning Representations, ICLR-23*, 2023.

Pietro Barbiero, Gabriele Ciravegna, Francesco Giannini, Pietro Lió, Marco Gori, and Stefano Melacci. Entropy-based logic explanations of neural networks. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI-22*, pp. 6046–6054, 2022.

Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

Andrew Cropper and Sebastijan Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022.

Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pp. 16–22. AAAI Press, 1998.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

Richard Evans, Matko Bosnjak, Lars Buesing, Kevin Ellis, David P. Reichert, Pushmeet Kohli, and Marek J. Sergot. Making sense of raw input. *Artif. Intell.*, 299:103521, 2021.

Maziar Moradi Fard, Thibaut Thonet, and Éric Gaussier. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognit. Lett.*, 138:185–192, 2020.

Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. Learning first-order rules with differentiable logic program semantics. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3008–3014, 2022a.

Kun Gao, Hanpin Wang, Yongzhi Cao, and Katsumi Inoue. Learning from interpretation transition using differentiable logic programming semantics. *Mach. Learn.*, 111(1):123–145, 2022b.

Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. A differentiable first-order rule learner for inductive logic programming. *Artif. Intell.*, 331:104108, 2024.

Yuanduo He, Xu Chu, Guangju Peng, Yasha Wang, Zhu Jin, and Xiaorong Wang. Mining rules from real-valued time series: A relative information-gain-based approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC-18*, pp. 388–397. IEEE Computer Society, 2018.

Céline Hocquette, Andreas Niskanen, Matti Järvisalo, and Andrew Cropper. Learning MDL logic programs from noisy data. In *Proceedings of the 39th Annual AAAI Conference on Artificial Intelligence, AAAI-24*, pp. 10553–10561. AAAI Press, 2024.

Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Mach. Learn.*, 94(1):51–79, 2014.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of the 5th International Conference on Learning Representations, ICLR-17*, 2017.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljacic, Thomas Y. Hou, and Max Tegmark. KAN: kolmogorov-arnold networks. *CoRR*, abs/2404.19756, 2024.

John W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS-18*, pp. 3753–3763, 2018.

Emanuele Marconato, Gianpaolo Bontempo, Elisa Ficarra, Simone Calderara, Andrea Passerini, and Stefano Teso. Neuro-symbolic continual learning: Knowledge, reasoning shortcuts and concept rehearsal. In *Proceedings of the 40th International Conference on Machine Learning, ICML-23*, volume 202, pp. 23915–23936. PMLR, 2023.

Eleonora Misino, Giuseppe Marra, and Emanuele Sansone. VAEL: bridging variational autoencoders and probabilistic logic programming. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS-22*, 2022.

Stephen H. Muggleton and Cao Feng. Efficient induction of logic programs. In *Algorithmic Learning Theory, First International Workshop, ALT-90*, pp. 368–381. Springer/Ohmsha, 1990.

Stephen H. Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

Stephen H. Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20 - biography and future challenges. *Mach. Learn.*, 86(1): 3–23, 2012.

Andrea Passerini, Paolo Frasconi, and Luc De Raedt. Kernels on prolog proof trees: Statistical learning in the ILP setting. *J. Mach. Learn. Res.*, 7:307–342, 2006.

Yin Jun Phua and Katsumi Inoue. Learning logic programs using neural networks by exploiting symbolic invariance. In *Proceedings of the 30th international conference on Inductive Logic Programming, ILP-21*, volume 13191 of *Lecture Notes in Computer Science*, pp. 203–218. Springer, 2021.

Yin Jun Phua and Katsumi Inoue. Variable assignment invariant neural networks for learning logic programs. In *Proceedings of the 18th International Conference on Neural-Symbolic Learning and Reasoning, NeSy-24*, volume 14979 of *Lecture Notes in Computer Science*, pp. 47–61. Springer, 2024.

J. Ross Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5:239–266, 1990.

Luc De Raedt and Luc Dehaspe. Clausal discovery. *Mach. Learn.*, 26(2-3):99–146, 1997.

Lior Rokach and Oded Maimon. Clustering methods. In *The Data Mining and Knowledge Discovery Handbook*, pp. 321–352. Springer, 2005.

Chiaki Sakama, Katsumi Inoue, and Taisuke Sato. Logic programming in tensor spaces. *Ann. Math. Artif. Intell.*, 89(12):1133–1153, 2021.

Emanuele Sansone. The triad of failure modes and a possible way out. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems on the 4th Workshop on Self-Supervised Learning: Theory and Practice (SSL, NeurIPS-23)*, 2023.

Emanuele Sansone and Robin Manhaeve. Learning symbolic representations through joint generative and discriminative training. In *Proceedings of the 11th International Conference on Learning Representations on Neurosymbolic Generative Models Workshops (NeSy-GeMs, ICLR-23)*, 2023.

Hikaru Shindo, Viktor Pfanschilling, Devendra Singh Dhami, and Kristian Kersting. αILP: thinking visual scenes as differentiable logic programs. *Mach. Learn.*, 112(5):1465–1497, 2023.

Sever Topan, David Rolnick, and Xujie Si. Techniques for symbol grounding with SATNet. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS-21*, pp. 20733–20744, 2021.

Bowen Wang, Liangzhi Li, Yuta Nakashima, and Hajime Nagahara. Learning bottleneck concepts in image classification. In *Proceedings of the 34th IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR-23*, pp. 10962–10971, 2023.

Haishuai Wang, Qin Zhang, Jia Wu, Shirui Pan, and Yixin Chen. Time series feature learning with labeled and unlabeled data. *Pattern Recognit.*, 89:55–66, 2019.

Jin Wang, Ping Liu, Mary Fenghua She, Saeid Nahavandi, and Abbas Z. Kouzani. Bag-of-words representation for biomedical time series classification. *Biomed. Signal Process. Control.*, 8(6): 634–644, 2013.

Zhao Xu and Koichi Funaya. Time series analysis with graph-based semi-supervised learning. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA-15*, pp. 1–6. IEEE, 2015.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NIPS-17*, pp. 2319–2328, 2017.

Eric Zhan, Jennifer J. Sun, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. Unsupervised learning of neurosymbolic encoders. *Trans. Mach. Learn. Res.*, 2022, 2022.

Han Zhang, Ian J. Goodfellow, Dimitris N. Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML-19*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7354–7363. PMLR, 2019.