# Learning Explainable Linguistic Expressions with Neural Inductive Logic Programming for Sentence Classification

**Prithviraj Sen**
IBM Research
San Jose, CA, USA
senp@us.ibm.com

**Marina Danilevsky**
IBM Research
San Jose, CA, USA
mdanile@us.ibm.com

**Yunyao Li**
IBM Research
San Jose, CA, USA
yunyaoli@us.ibm.com

**Siddhartha Brahma**
Google Research
Mountain View, CA, USA
sidbrahma@google.com

**Matthias Boehm**
Graz University of Technology
Graz, Austria
m.boehm@tugraz.at

**Laura Chiticariu**
IBM Watson
San Jose, CA, USA
chiti@us.ibm.com

**Rajasekar Krishnamurthy**
IBM Watson
San Jose, CA, USA
rajase@us.ibm.com

## Abstract

Interpretability of predictive models is becoming increasingly important with growing adoption in the real-world. We present RuleNN, a neural network architecture for learning transparent models for sentence classification. The models are in the form of rules expressed in first-order logic, a dialect with well-defined, human-understandable semantics. More precisely, RuleNN learns linguistic expressions (LE) built on top of predicates extracted using shallow natural language understanding. Our experimental results show that RuleNN outperforms statistical relational learning and other neuro-symbolic methods, and performs comparably with black-box recurrent neural networks. Our user studies confirm that the learned LEs are explainable and capture domain semantics. Moreover, allowing domain experts to modify LEs and instill more domain knowledge leads to human-machine co-creation of models with better performance.

## 1 Introduction

Difficult-to-interpret, black-box predictive models have been shown to harbor undesirable biases (e.g., racial bias in computing risk of recidivism among criminals (Angwin et al., 2016; Liptak, 2017)). Renewed interest in interpretability (BlackBoxNLP) has led to techniques for explaining not only the inner workings of the model but also to explain how it derives a prediction.

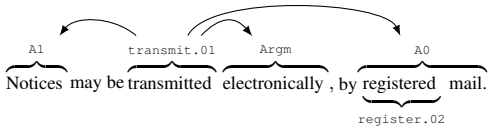
While various techniques for explainability exist (see survey by Guidotti et al. (2018)), one popular approach explains predictions from a black-box model by using a surrogate models (Ribeiro et al., 2016). Another extracts explanations from neural network layer activations, especially when said activations appeal to human intuition such as attention (Bahdanau et al., 2015) which may be interpreted as importance weights assigned to (latent) features derived by the model. While such approaches are useful, they raise questions such as whether the purported explanation provided by the surrogate correctly reflects the process employed by the black-box model to arrive at the prediction (sometimes called inexact explanation (Chu et al., 2018)). Similarly, attention only provides noisy explanations (Serrano and Smith, 2019). Such approaches leave room for improvement because explainability is treated as an after-thought whereas our goal is to treat it as a first-class citizen. In other words, is it possible to devise a neural network that directly learns a model expressed in a clear, human-readable dialect?

*First-order logic* (FOL) is a human-interpretable fragment of logic that includes existential ($\exists$) and universal ($\forall$) quantification along with propositional logic's conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) operators. FOL can model diverse applications such as data integration (Singla and Domingos, 2006), image interpretation (Donadello et al., 2017), and as the next example shows, natural language processing (NLP).

Consider identifying sentences denoting communication between two parties in a legal contract.



*Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 4211–4221,
November 16–20, 2020. 




$$\texttt{communication(s)} \leftarrow \texttt{Contains}(s, a) \wedge a.\texttt{A1} = \textit{notice} \wedge (a.\text{verb} = \textit{inform} \vee a.\text{verb} = \textit{transmit})$$

Figure 1: Legal contract sentence and an LE for label `communication` (syntax simplified for brevity).

Figure 1 shows such a sentence along with linguistic abstractions in PropBank notation (Palmer et al., 2005) extracted using shallow semantic parsing. It consists of two actions, `transmit.01` and `register.02`, with arguments `A0`, `A1` and `Argm`. Figure 1 also shows an FOL rule that assigns label `communication` by evaluating linguistic clues. We refer to such rules as linguistic expressions (LE). More precisely, the LE in the figure assigns `communication` if: action belongs to the sentence, surface form of the action belongs to a dictionary containing "inform" and "transmit", and its `A1` argument matches a dictionary containing "notice". Not only does the LE's conditions evaluate to true on the sentence example, but attribution, i.e., which parts of the sentence led to the prediction is also clear: `transmit.01`, rather than `register.02`, leads to the predicted label because the surface form corresponding to `transmit.01`'s action and its `A1` argument is "transmit" and "notice", respectively. This allows domain experts to verify the LE's semantics and explain predictions without encountering aforementioned complications due to the use of surrogate models, for instance.

FOL rules may be learned using inductive logic programming (ILP) (Muggleton, 1996), statistical relational learning (StarAI) (Getoor and Taskar, 2007) or neuro-symbolic AI (Evans and Grefenstette, 2018). None of these however, target NLP. FOL rules consist of predicates which are Boolean functions that specify conditions for the rule to hold, e.g., `Contains` and dictionary-match conditions in Figure 1. Thus, we need to: 1) learn discriminative predicates, 2) combine them into LEs, and 3) learn multiple LEs in case one is insufficient. Figure 2 summarizes our approach. We leverage natural language understanding (NLU) to generate well defined, human-interpretable predicates. Our main contribution is RuleNN, a neural

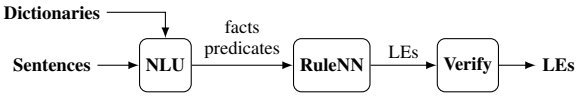



Figure 2: Pictorial depiction of our approach.

network (NN) comprising *predicate generation* (PGM) and *clause generation* (CGM) modules for learning and combining discriminative predicates to form LEs. By adding more modules, RuleNN can learn multiple LEs jointly. We also show how to extract LEs expressed in crisp FOL from RuleNN post-hoc that may, in turn, be handed to domain experts for verification and even modification, to instill further domain expertise going beyond the available training data.

By evaluating on two real-world sentence classification datasets and comparing against a host of baselines, we show that LEs learned by RuleNN lead to large gains in terms of area under the precision-recall curve (AUC-PR). Averaging across labels, RuleNN's AUC-PR is $6.8\times$, $7.6\times$, $1.5\times$ that of ILP, StarAI, other neuro-symbolic AI approaches, respectively. We also compare against black-box methods that are far less explainable. In particular, we show that RuleNN's LEs are comparable to bi-directional LSTMs (Hochreiter and Schmidhuber, 1997) with GloVE embeddings (Pennington et al., 2014). A user study with 4 domain experts confirms that RuleNN's LEs are interpretable, capture domain semantics and are conducive to human-machine model co-creation. We make the following contributions:

- Propose LEs for explainable NLP constructed using predicates from NLU.
- Propose RuleNN, a modular NN for learning multiple LEs. Given predicates, RuleNN can learn rules for any application, not just NLP.
- Compare with ILP, StarAI, neuro-symbolic AI and LSTMs on real sentence classification data.
- Evaluate explainability of LEs via a user study.
- Illustrate human-machine co-creation by showing how humans interact with explainable LEs.

## 2 Related Work

Inductive logic programming (ILP) (Muggleton, 1996) learns rules that perfectly entail the positive examples and reject all negatives. Top-down ILP systems (Muggleton et al., 2008; Corapi et al., 2010; Cropper and Muggleton, 2015) in particular,

generate rules before testing them on data. Since a 0-error rule may not exist, noise-tolerant ILP (Muggleton et al., 2018) learns rules that minimize error which is more suited for noisy real-world scenarios. We compare RuleNN against top-down and noise-tolerant ILP in Section 5.

Markov logic network (MLN) (Richardson and Domingos, 2006), a member of statistical relational learning (StarAI) (Getoor and Taskar, 2007), comprises weighted rules to extend Markov random fields (Pearl, 1988) to the first-order setting. A long line of work exploring various techniques culminated in the LSM heuristic (Kok and Domingos, 2010) that learns MLN rules before estimating parameters. Since such a stepwise approach can be computationally expensive, BoostSRL (Khot et al., 2011) jointly learns rules and parameters by approximating the gradient using functional gradient boosting (Friedman, 2001). RuleNN replaces logical operations with differentiable functions, thus learning LEs end-to-end without approximations. Section 5 reports results of LSM and BoostSRL.

Neuro-symbolic AI employs neural networks for rule induction (Yang et al., 2017; Evans and Grefenstette, 2018; Dong et al., 2019). To the best of our knowledge however, none of these address NLP tasks such as sentence classification. NeuralLP (Yang et al., 2017) for instance, learns restricted chain rules to predict links in knowledge graphs. ∂ILP (Evans and Grefenstette, 2018) learns recursion by materializing all possible logic programs thus incurring exponential complexity. Neural Logic Machines (Dong et al., 2019) translate FOL to tensor operations and multi-layer perceptrons thus precluding extraction of FOL rules. In contrast, we learn LEs for NLP by constructing predicates from NLU, RuleNN's complexity is proportional to the number of rules, and we also extract FOL rules once training has converged. Other combinations of FOL and neural networks include Kazemi and Poole (2018); Sourek et al. (2018); Donadello et al. (2017); Gupta et al. (2020) and neural theorem provers (Rocktäschel and Riedel, 2017), which convert user-specified rules into neural networks. In particular, neural module networks (Gupta et al., 2020) convert compositional questions into modules in a neural network with the goal being to learn all parameters jointly. In contrast to RuleNN, none of these directly learn rules from labeled data.

Recall that in Figure 1, the sentence comprises actions and we evaluate the LE on each action. This is in fact multiple instance learning (MIL) (Dietterich et al., 1997; Amores, 2013) where one instance (e.g., sentence) contains a set of instances (e.g., actions) and is strictly more general than independent and identically distributed (IID) classification. Previous use of MIL includes aspect-based sentiment analysis (Pappas and Popescu-Belis, 2014). In Section 5, we compare RuleNN against MIL classifiers including MIRI (Bjerring and Frank, 2011), a MIL rule-learner, and MITI (Blockeel et al., 2005), a MIL decision tree learner.

## 3 Constructing Predicates based on Natural Language Understanding

In this section, we describe how to define human interpretable predicates by leveraging semantic role labeling (SRL) (Jurafsky and Martin, 2014) and syntactic parsing. In Section 4, we show how to learn discriminative LEs on top of such predicates.

### 3.1 From Linguistic Features to Predicates

We begin by introducing first-order logic constructs such as logical predicates, constants and facts before applying them to the NLP domain. We ground all definitions via examples subsequently.

**Definition 3.1** (Logical Predicate)**.** *A* predicate *is a Boolean-valued function returning true* (1) *or false* (0)*. Formally, let* $x$ *denote a logical variable that takes values from domain of constants* $Dom$*. Then* Pred$(x_1, \dots x_n)$ *denotes an* $n$*-ary* predicate *where* $x_i$ *is either a logical variable or constant* $\in Dom$*.*

Pred$(x_1, \dots x_n)$ denotes a *ground atom* that evaluates to either true or false if $x_i$ denotes a constant, $\forall i = 1, \dots n$. A *fact* is an atom that holds true.

Given our interest in sentence classification, we will be dealing with logical constants corresponding to sentences and actions extracted from them. We describe two ways to generate predicates for NLP, the first of which uses SRL arguments and dictionaries.

**Definition 3.2** (SRL Predicate)**.** *Given SRL argument* $SemAttr$ *and dictionary* $h$*, predicate* $\texttt{SemAttrMatches}_\texttt{h}(a)$ *is true if* $h$ *contains action* $a$*'s surface form corresponding to* $SemAttr$*.*

Let $h = \{w_1, \dots w_d\}$ denote a dictionary of semantically related words (or phrases) such that $w_i \in \mathcal{V}$, $\forall i = 1, \dots d$, where $\mathcal{V}$ denotes the vocabulary. Also, let $\mathcal{H}$ denote a set of dictionaries such that $\forall h \in \mathcal{H} : h \subseteq \mathcal{V}$ and let $SemAttr$ denote an

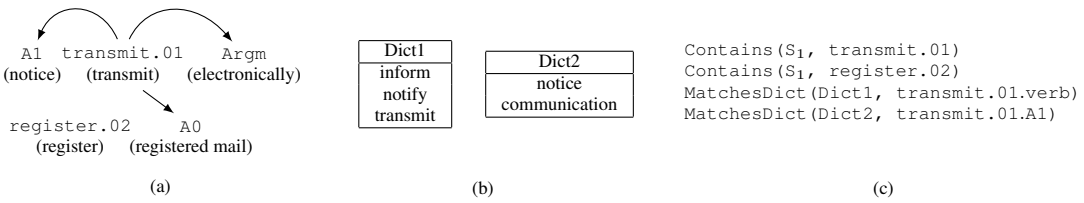



Figure 3: Generating (c) facts and predicates from (a) shallow semantic parsing and (b) dictionaries.

argument for an action extracted using SRL. Using Definition 3.2, one can define one predicate[1] per SRL argument per dictionary $h \in \mathcal{H}$.

**Definition 3.3** (Syntactic Predicate)**.** *Given syntactic argument $SynAttr$ and $v \in Dom(SynAttr)$, predicate $IsSynAttr_v(a)$ holds true if action $a$'s value of $SynAttr$ is $v$.*

The second type of predicate we propose is derived from syntactic arguments of actions such as tense and voice. Let $SynAttr$ denote such an argument whose domain is denoted by $Dom(SynAttr)$ (e.g., $Dom(\text{voice}) = \{\text{active}, \text{passive}\}$). Definition 3.3 then allows creation of one predicate per domain value $v$ per syntactic argument.

The main construct in FOL is a rule or *clause*:

$$R : c \leftarrow b_1, b_2, \ldots b_n$$

where $R$ is an identifier, $b_1, \ldots b_n$ denote predicates in its *body* and $c$ denotes the *head* predicate. *If* the body is true *then* the head is also true, which in the context of classification, will be the label predicate $\ell(x)$ which in turn, if true, implies that the instance denoted by $x$ is to be assigned the label. In short, we treat clauses as binary classifiers. For multiple labels, we utilize distinct label predicates and learn distinct clauses.

**Definition 3.4** (Linguistic Expression or LE)**.** *A clause defined over logical variables representing sentence* s *and action* a*, includes a distinguished binary predicate* `Contains`*(s,a) which is true if* a *belongs to* s*, and whose body contains SRL and/or syntactic predicates (see Definitions 3.2 and 3.3).*

Figure 3 (a) shows actions and their arguments extracted from the example in Figure 1 expressed in PropBank annotation schema (Palmer et al., 2005). One such action is `transmit.01`, whose SRL arguments include *notice* (`A1`, the target of the action), *electronically* (`ARGM`, how the action is performed.) and *registered mail* (`A0`, the agent

[1]"Predicate" refers to the logical kind used to build LEs, not to be confused with SRL's predicate-argument structure.

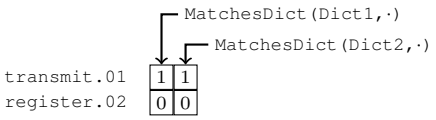



Figure 4: Example predicate matrix where rows and columns denote actions and predicates, respectively.

of the action). We use shorthand $a.SemAttr$ to refer to action $a$'s surface form associated with $SemAttr$. Figure 3 (b) and (c) show two dictionaries and the facts generated using these, respectively. Dict1 contains three tokens, one of which matches the surface form of `transmit.01`, i.e. `transmit.01.verb` = *transmit*. This leads to the fact `MatchesDict`(Dict1, `transmit.01.verb`) which is syntactic sugar for the SRL predicate produced with $SemAttr$ verb and dictionary Dict1 in accordance with Definition 3.2. Similarly, `transmit.01.A1` = *notice* and dictionary Dict2 leads to another fact `MatchesDict`(Dict2, `transmit.01.A1`).

### 3.2 Problem Formulation

Definition 3.4 states that an LE contains two kinds of logical variables and `Contains` is the only predicate defined on sentence $s$ (neither Definition 3.2 nor 3.3 introduce a predicate over $s$). This implies that we need to predict sentence labels based on the facts defined over actions. Indeed, the LE's body in Figure 1 contains free variable $a$ not appearing in the head and under standard existential semantics this implies that the head is true only if there exists an $a$ which satisfies the body.

Let $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ denote a binary class, labeled sample where $x_i$ is a constant denoting an instance to be classified (in our case, sentences) with label $y_i \in \{0, 1\}$. Let $\mathcal{R}_i$ denote $x_i$'s constituents (in our case, the set of actions from $x_i$). Also, let $\mathcal{P} = \{\text{Pred}_1, \ldots \text{Pred}_N\}$ denote unary predicates defined using Definitions 3.2 and 3.3. Since all predicates are unary, we can represent all facts associated with $x_i$ using a *predicate*

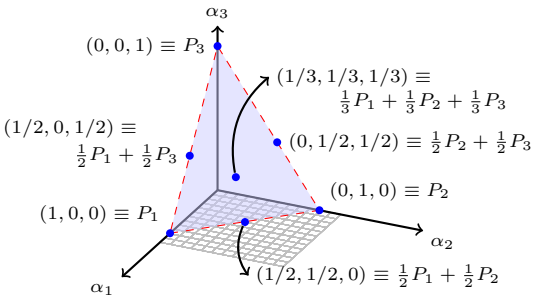



Figure 5: Convex hull of $\{P_1, P_2, P_3\}$. Highlighted points (in blue) denote specific predicate combinations.

*matrix* $M \in \{0,1\}^{|\mathcal{R}_i| \times |\mathcal{P}|}$ such that $M_{kj} = 1$ if $\text{Pred}_j(a)$ is true and 0 otherwise, where $a$ denotes the $k^{\text{th}}$ action in $\mathcal{R}_i$. Figure 4 shows an example of a predicate matrix. Our task is to learn (possibly, multiple) LEs that capture $x_i$'s label $y_i$.

## 4 RuleNN: Learning Linguistic Expressions with Neural Networks

We now present RuleNN, a neural network for learning LEs from labeled data. Since LEs are discrete objects, we first present a parameterized predicate that is defined in terms of learnable parameters. Subsequently, we introduce predicate and clause generation modules. By adding more of these modules, the architecture scales to facilitate learning of multiple, longer LEs.

### 4.1 Parameterized Predicate and Predicate Generation Module

Parameterized predicate (PP) expresses a linear (more precisely, convex) combination of predicates from $\mathcal{P}$. Consider $\mathcal{P}$'s convex hull which is the smallest convex set containing all its predicates. Since $P(a) \in \{0,1\}, \forall P \in \mathcal{P}$ (Definition 3.1), any point in the convex hull may be expressed as:

$$\text{PP}_{\mathcal{P}}(a;\boldsymbol{\alpha}) = \textstyle\sum_{P_i \in \mathcal{P}} \alpha_i P_i(a),\ \forall a \in R_i,\ x_i \in \mathcal{D}$$
$$\text{such that } \textstyle\sum_i \alpha_i = 1,\ \alpha_i \geq 0\ \forall i = 1, \ldots |\mathcal{P}|$$

Given $\boldsymbol{\alpha} = [\alpha_1, \ldots \alpha_{|\mathcal{P}|}]^\top$, *parameterized predicate* $\text{PP}_{\mathcal{P}}(a;\boldsymbol{\alpha})$ returns a distinct predicate combination of $\mathcal{P}$. In particular, when $\boldsymbol{\alpha}$ is a one-hot encoding $\text{PP}_{\mathcal{P}}(a;\boldsymbol{\alpha})$ results in a distinct predicate in $\mathcal{P}$ which corresponds to a corner of the convex hull. In fact, the hull spans a $(|\mathcal{P}|-1)$-simplex. Figure 5 shows the convex hull of 3 predicates that forms a 2-simplex or triangle and points in blue highlight salient predicate combinations.

Given an update scheme, e.g. backpropagation, $\text{PP}_{\mathcal{P}}(a;\boldsymbol{\alpha})$ can switch from one predicate (combination) to another by updating $\boldsymbol{\alpha}$. Thus, PPs enable learning LEs via gradient-based optimization. Figure 6 (a) depicts PP as a combination of layers or a *predicate generation module* (PGM). To enforce the non-negativity and summation constraints, we derive $\boldsymbol{\alpha}$ (shown in blue) from auxiliary variables $\boldsymbol{\gamma} \in \mathbb{R}^{|\mathcal{P}|}$ (shown in green) using the `softmax` transform (Srivastava and Sutton, 2017):

$$\alpha_i = \frac{e^{\gamma_i}}{\sum_{j=1}^{|\mathcal{P}|} e^{\gamma_j}},\ \forall i = 1, \ldots |\mathcal{P}|$$

$\text{PP}_{\mathcal{P}}(a;\boldsymbol{\alpha})$ (vector in red in Figure 6 (a)) is then given by the matrix-vector product $M\boldsymbol{\alpha}$ where $M$ denotes the predicate matrix of some $x_i \in \mathcal{D}$.

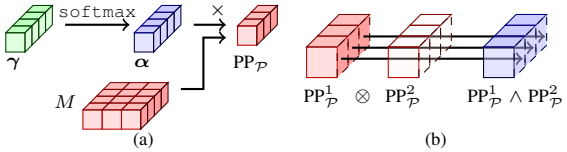



Figure 6: (a) Predicate Generation Module. (b) Conjunction using Hadamard product.

### 4.2 Clause Generation Module

Figure 1's LE is a conjunction of two SRL predicates; such combinations can be learned by combining multiple PGMs into a *clause generation module* (CGM). Let $m$ denote the number of predicates in an LE. Following other works (Yang et al., 2017; Sourek et al., 2018), CGMs replace non-differentiable logical conjunction with a smooth $t$-norm operator (Esteva and Godo, 2001). While various $t$-norms exist, product $t$-norm leads to better results (Evans and Grefenstette, 2018). Note that, RuleNN's architecture does not rely on product $t$-norm and may easily switch to another $t$-norm if desired. Product $t$-norm of $\text{PP}^1_{\mathcal{P}}(a;\boldsymbol{\alpha}_1)$, $\text{PP}^2_{\mathcal{P}}(a;\boldsymbol{\alpha}_2)$ is given by Hadamard product or element-wise multiplication $\text{PP}^1_{\mathcal{P}}(a;\boldsymbol{\alpha}_1) \otimes \text{PP}^2_{\mathcal{P}}(a;\boldsymbol{\alpha}_2)$, $\forall a \in \mathcal{R}_i$, $\forall x_i \in \mathcal{D}$ (see Figure 6 (b)).

### 4.3 RuleNN Architecture

Let $k$ denote the desired number of LEs to learn. Figure 7 depicts RuleNN consisting of $k$ CGMs shown in solid boxes (corresponding to $k$ LEs), consisting of $m$ PGMs per CGM shown in dashed boxes (corresponding to $m$ predicates within each LE). Our motivation for learning multiple LEs is because a single rule may not result in high precision *and* high recall due to the precision-recall trade-off. Thus, learning a disjunction of more than

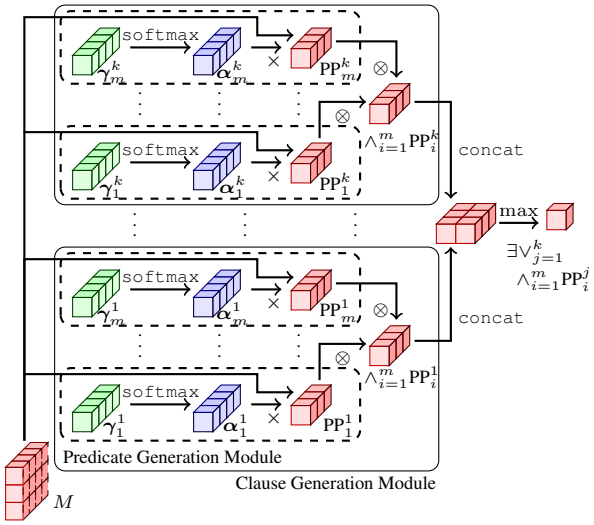



Figure 7: RuleNN for learning $k$ $m$-length clauses.

| Name | Global/Local | Description |
|---|---|---|
| $M$ | Local | Predicate matrix for instance |
| $\mathcal{R}$ | Local | Actions in instance |
| $PP_i^j$ | Local | Responses for actions belonging to instance |
| $\alpha_i^j$ | Global | Attention weights defining a learned predicate |
| $\gamma_i^j$ | Global | Log attention weights for learned predicate |
| $k$ | Global | Number of LEs (hyperparameter) |
| $m$ | Global | Length of LEs (hyperparameter) |

Table 1: Notation with description

one LE, i.e., the label is assigned if any LE holds true for the sentence, can lead to improved results.

In detail, the bottom left of Figure 7 shows a predicate matrix $M$ for some $x \in \mathcal{D}$ whose constituent actions are given by $\mathcal{R}$. To model conjunction, $j^{\text{th}}$ CGM's output is computed by element-wise multiplying $|\mathcal{R}|$-dimensional vectors $\text{PP}_1^j, \ldots \text{PP}_m^j$ produced by corresponding PGMs, where (superscript) subscript denote index of (CGM) PGM. Given outputs of the CGMs, 2 operations remain: 1) existential over actions following the semantics of the LE, and 2) disjunction over all LEs. We treat existential as a disjunction-like operator. Since logical disjunction is also not differentiable, neuro-symbolic AI replaces it with a $t$-conorm such as $\max$ (Dong et al., 2019) whose (sub)gradient is available. The (scalar) $\max$ across all CGM outputs models both the existential and disjunction to return a score for $x$ that may be compared to its label $y$ in $\mathcal{D}$ via a loss function. Table 1 describes our notation for easy reference.

### 4.4 Further Optimizations and LE Retrieval

While RuleNN supports learning $m$ PPs per CGM, if $\exists i \neq i'$ such that $\boldsymbol{\alpha}_i^j = \boldsymbol{\alpha}_{i'}^j$, then $j^{\text{th}}$ CGM consists of $< m$ distinct PPs. Effectively, RuleNN

**Algorithm 1:** Post-hoc LE retrieval

**input :** Learned $\alpha_1, \ldots \alpha_m$ and training data $\mathcal{D}$.
**output :** List of LEs.

1 $S \leftarrow \{\}$ `// Loop goes over` $\binom{|\mathcal{P}|}{m}$ `combinations`
2 **while** more predicate combinations exist **do**
3 $\quad (p_1, \ldots p_m) \leftarrow$ get next predicate combination
4 $\quad$ **if** $\prod_{i=1}^m \alpha_{ip_i} > 0 \wedge \exists x \in \mathcal{D}$ such that $(p_1, \ldots p_m) \sim x$ **then** $S \leftarrow S \cup \{(p_1, \ldots p_m)\}$
5 **return** $S$

| | TREC | Contracts |
|---|---|---|
| #Sentences/Questions (Train) | 5301 | 28174 |
| #Sentences/Questions (Test) | 497 | 1259 |
| #Labels | 6 | 9 |
| #Actions (Train) | 6996 | 105552 |
| #Actions (Test) | 562 | 4850 |
| #Actions per sentence | 1.3 | 3.75 |

Table 2: Broad-level dataset statistics

learns $k$ LEs containing *up to* $m$ PPs each. To handle class skew, i.e., $\mathcal{D}$ consists of more negative than positive examples, we utilize negative sampling (Mikolov et al., 2013). We also apply dropout (Srivastava et al., 2014) just before max-pooling to zero-out outputs from randomly chosen CGMs. Once learning has converged, we can use Algorithm 1 to retrieve LEs expressed in FOL. Given $\boldsymbol{\alpha}_1, \ldots \boldsymbol{\alpha}_m$ learned from a single CGM, Algorithm 1 considers each $m$-combination of predicates from $\mathcal{P}$ and returns it as an LE if (Line 4): 1) its associated weight (product of corresponding numbers in $\boldsymbol{\alpha}_i$, $\forall i = 1, \ldots m$) is non-zero, and 2) it evaluates to true on some instance in $\mathcal{D}$. When learning $k$ CGMs, we invoke Algorithm 1 once per CGM and union the LEs. Algorithm 1's complexity is exponential in $m$ but it is efficient for short LEs which makes sense since longer LEs are hard to interpret. In practice, post-hoc retrieval results in a few hundred LEs (Section 5 discusses how to navigate such a set of LEs).

## 5 Experiments

**Datasets**: We experiment with two datasets: TREC (Li and Roth, 2002) comprising questions, and the real-world Contracts data (proprietary) comprising sentences from legal contracts among enterprises. Contracts calls for out-of-domain generalization since its training set involves contracts with IBM as first party while the test set includes more diverse companies. Table 2 provides broad-level statistics. Sentences in Contracts may be labeled with 0, 1 or more labels (multi-label classification), so we treat each label as a binary class labeling task. Table 3

| Label | Skew | $\|\mathcal{P}\|$ |
|---|---|---|
| W | 0.09 | 101 |
| SoW | 0.07 | 48 |
| DR | 0.06 | 80 |
| IP | 0.05 | 79 |
| C | 0.06 | 39 |
| P&T | 0.10 | 117 |
| T&T | 0.08 | 77 |
| P&B | 0.05 | 95 |
| L | 0.04 | 71 |

(a) Contracts: Label statistics

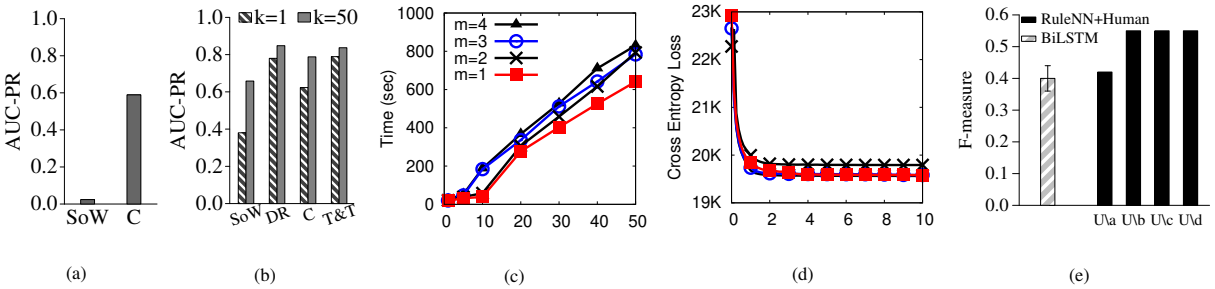
| Label | Skew | $\|\mathcal{P}\|$ |
|---|---|---|
| LOC | 0.18 | 133 |
| HUM | 0.28 | 109 |
| NUM | 0.17 | 127 |
| ENTY | 0.22 | 137 |
| DESC | 0.22 | 122 |
| ABBR | 0.02 | 38 |

(b) TREC: Label Statistics

| | | Predicate-based | | | | | | | (↓ours↓) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Label | MG□ | $MG_{NT}$□ | MITI□ | MIRI□ | MINet■ | LSM□ | BSRL□ | NeuralLP□ | *RuleNN*□ | BiLSTM■ |
| CONTRACTS | W | *NR* | 0.07 | 0.184 | 0.156 | 0.294 | – | 0.183 | 0.537 | 0.685 | **0.805 ± 0.010** |
| | SoW | *NR* | *NR* | 0.011 | 0.011 | 0.018 | – | 0.015 | 0.438 | 0.658 | **0.689 ± 0.030** |
| | DR | *NR* | 0.05 | 0.144 | 0.147 | 0.258 | – | 0.021 | 0.614 | **0.848** | 0.807 ± 0.030 |
| | IP | *NR* | 0.13 | 0.145 | 0.153 | 0.244 | – | 0.148 | 0.550 | **0.844** | 0.787 ± 0.050 |
| | C | *NR* | 0.38 | 0.157 | 0.149 | 0.580 | – | 0.545 | 0.574 | **0.788** | 0.653 ± 0.020 |
| | P&T | *NR* | 0.30 | 0.111 | 0.083 | 0.314 | – | 0.269 | 0.516 | 0.813 | **0.802 ± 0.030** |
| | T&T | *NR* | 0.11 | 0.406 | 0.372 | 0.586 | – | 0.591 | 0.560 | 0.837 | **0.846 ± 0.020** |
| | P&B | *NR* | 0.10 | 0.111 | 0.122 | 0.154 | – | 0.192 | 0.533 | **0.819** | 0.786 ± 0.010 |
| | L | *NR* | 0.07 | 0.115 | 0.126 | 0.12 | – | 0.205 | 0.464 | 0.750 | **0.741 ± 0.070** |
| TREC | LOC | *NR* | *NR* | 0.710 | 0.699 | 0.833 | 0.473 | 0.835 | 0.470 | 0.904 | **0.998 ± 0.001** |
| | HUM | *NR* | 0.36 | 0.771 | 0.770 | 0.922 | 0.565 | 0.927 | 0.558 | 0.912 | **0.999 ± 0.000** |
| | NUM | *NR* | *NR* | 0.687 | 0.680 | 0.821 | 0.497 | 0.756 | 0.497 | 0.856 | **0.996 ± 0.004** |
| | ENTY | *NR* | *NR* | 0.365 | 0.373 | 0.591 | 0.481 | 0.425 | 0.576 | 0.745 | **0.957 ± 0.020** |
| | DESC | *NR* | 0.52 | 0.331 | 0.334 | 0.540 | 0.498 | 0.519 | 0.437 | 0.789 | **0.995 ± 0.003** |
| | ABBR | *NR* | *NR* | 0.731 | 0.731 | 0.688 | 0.542 | 0.735 | 0.443 | 0.774 | **1.000 ± 0.000** |

(c) *NR* and – denote no LEs learned and non-convergence, resp. Bold-font and underscore denotes best performing approach and best predicate-based method, resp. RuleNN (ours) is the best predicate-based method. Variation of BiLSTM's AUC-PR due to changing hidden dimensions is shown after ±.

Table 3: Dataset statistics and AUC-PR results




(a) (b) (c) (d) (e)

Figure 8: (a) ∂ILP results. (b) Quality gains due to increasing $k$. (c) RuleNN per-epoch time vs. $k$ (x-axis) and $m$. (d) Convergence vs. epochs (x-axis). (e) Domain experts aid generalization where $U$ denotes all users $\{a, b, c, d\}$.

(a) lists per-label[2] class skew, which is defined as the ratio of positive sentences divided by negatives. Note that, C denotes Communication (the label in the running example in Figure 1). Each sentence is processed using SystemT's (Krishnamurthy et al., 2008) SRL and dependency parser. We extract each action's tense, aspect, mood, modalclass, voice and polarity (syntactic arguments), and also semantic arguments such as `A0`, `A1`, `Argm` etc. Table 3 (a) lists the number of predicates constructed using hand-crafted dictionaries for each label following the process described in Section 3. We use TREC's standard train/test split to aid comparison which also exhibits significant class skew (Table 3 (b)), automatically construct dictionaries by capturing surface forms (from the training set) that discriminate well among its labels and construct predicates by extracting the same syntactic and semantic arguments stated previously.

**Methods Compared**: RuleNN learns $k$=50 LEs containing up to $m$=4 predicates. We set dropout=0.5, batchsize=64, stepsize=0.01 and use SGD with momentum=0.9. We compare against NeuralLP and ∂ILP, from neuro-symbolic AI; LSM and BoostSRL (BSRL), from StarAI; MITI and MIRI, from multiple instance learning; and top-down ILP system metagol (MG) (Cropper and Muggleton, 2015) and its noise-tolerant variant $\text{Metagol}_{NT}$ ($\text{MG}_{\text{NT}}$) (Muggleton et al., 2018). All of these are described in Section 2, and learn rules which we denote by □ (aka "white-box" method). We also compare with black-box methods (denoted by ■): MINet (Wang et al., 2018) and recurrent neural networks. MINet achieves MIL using a deep neural net with fully connected layers. BiLSTM replaces tokens in the sentence with GloVe embeddings (Pennington et al., 2014) followed by a bi-directional LSTM whose hidden vectors are aggregated to produce the label. By varying hidden units from 200 to 600, we obtain a range for BiLSTM performance. Note that, only BiLSTM gets raw sentence tokens as input. MINet along with □ methods, receive the same predicates and dictionaries that RuleNN receives as input. Thus, we refer

[2]Full label names are available (Legal Categories).

$R_2$ :communication($s$) ← Contains($s, a$)
∧ ($a$.Argm ∈ {*immediately, electronically,..*})
∧ ($a$.verb ∈ {*notify, inform,..*})

notify.01 Argm
$S_2$ : Notify buyer immediately upon completion or termination of any assignment and return buyer's identification badge.

---

$R_3$ :communication($s$) ← Contains($s, a$)
∧ ($a$.A0 contains *notice*
∨ $a$.A0 contains *communication*)
∧ $a$.tense = future

A0 be.01
$S_3$ : *Notices* required in writing under this agreement will be made to the appropriate contact(s) ..

---

$R_4$ :communication($s$) ← Contains($s, a$)
∧ $a$.voice = passive
∧ ($a$.A1 contains *notice*
∨ $a$.A1 contains *communication*)

A1 provide.01
$S_4$ : All *notices*, with the exception of legal notices, may also be provided by facsimile.

Figure 9: LEs learned by RuleNN with a sentence example each on which they hold true.

$R_5$ :term-&-termination($s$) ← Contains($s, a$)
∧ $a$.aspect = simple ∧ ($a$.verb ∈ {*terminate, expire, cease*})

---

$R_6$ :liability($s$) ← Contains($s, a$) ∧ $a$.aspect = simple
∧ ($a$.A1 contains *liable* ∨ $a$.A1 contains *liability*)

Figure 10: LEs learned by RuleNN for T&T and L.

to them as predicate-based methods.

### 5.1 Comparative Results

Table 3 (c) and Figure 8 (a) report area under precision-recall curve (AUC-PR) for all methods.

**Vs. predicate-based and □ methods**: It is unlikely that there exists any LE which perfectly entails the positives and rejects all negative examples. Thus, MG learns no rules (NR) and $MG_{NT}$'s AUC-PR is poor. Both LSM and BSRL are susceptible to class skew (Khot et al., 2011). Despite running for 5 days, LSM did not provide a result (denoted by -) on Contracts, the larger of the 2 datasets. NN-based MINet outperforms MITI and MIRI. RuleNN shows impressive performance despite class skew and scale. Averaged across labels, it outperforms MINet by 16% on TREC and 595% on Contracts. Among □ methods, RuleNN outperforms BSRL by 25% on TREC and 109% on Contracts.

**Vs. neural-symbolic methods**: NeuralLP learns chain rules (also called closed paths) for link prediction which differs from LE structure defined in Section 3. RuleNN outperforms it by 69% on TREC and 48% on Contracts. ∂ILP's prohibitive complexity, $O(|\mathcal{P}|^{mk})$, prevents us from learning more than 2 LEs containing 2 predicates only (settings from Evans and Grefenstette (2018)). Its AUC-PR (Figure 8 (a)) on C and SoW, Contracts' labels with fewest predicates (Table 3 (a)), is erratic. RuleNN outperforms it on both labels.

**Vs. ■ methods**: RuleNN outperforming MINet shows that despite learning explainable LEs, it can still improve over black-box methods. In its smallest setting, BiLSTM contains (upwards of) 400000 learnable parameters (with 300-dim. GloVe embeddings). In contrast to RuleNN's $O(|\mathcal{P}|mk)$ parameters (and its settings specified earlier in this section), BiLSTM's parameter set is an order of magnitude larger. This allows BiLSTM to provide excellent results on shorter questions in TREC (containing 1.3 actions on avg., Table 2) but overfits on Contracts, there being marked differences between data distributions of the training and test set. RuleNN's AUC-PR is comparable to BiLSTM's on Contracts. In fact, it outperforms BiLSTM in 4 of 9 labels. This is in addition to the explainability offered by the learned LEs (we show examples subsequently).

**Impact of parameters and initialization**: Figure 8 (c) shows per-epoch training times for RuleNN on label C in Contracts against varying $k$ (x-axis) and $m$. RuleNN's runtime depends linearly on $k$ and is not affected much by $m$. This is a clear win against more expensive approaches such as ∂ILP and LSM. Increasing $k$ or $m$ may also allow a better fit. Figure 8 (b) shows which of Contracts' labels benefit by increasing $k$ from 1 to 50. We also tested different random initialization of $\gamma_i^j$, $\forall i, j$. RuleNN converges to similar cross-entropy loss across different initializations (Figure 8 (d)).

### 5.2 Explainability of Learned LEs

Algorithm 1 produces 188 LEs for C in Contracts. While this may seem excessive, it is possible to build a graphical user interface (GUI) (Yang et al., 2019) that allows for instance, to filter LEs based on a precision threshold and rank LEs based on say, recall, to efficiently navigate through such a set of LEs. Unfortunately, we know of no objective met-

ric that can be automatically computed to directly measure LE's explainability. Thus, we contrast example LEs learned by RuleNN with BSRL's, one of the better performing rule-learning baselines.

Besides the LE in Figure 1, Figure 9 shows 3 more RuleNN LEs with example sentences where $a.SemAttr$ `contains` $w$ is true when $a.SemAttr$ includes token $w$ which is useful when SRL extracts an extended piece of the sentence as $a.SemAttr$[3] (e.g., `A0` in $S_3$ in Figure 9). Clearly, all involved terms are `Communication`-specific. For instance, $S_2$ shows how verb *notify* appears in a sentence implying communication thus $R_2$ including it in its SRL predicate makes sense. Besides SRL predicates, these LEs also include syntactic predicates capturing the fact that legal contracts are often written in passive voice ($R_4$) or future tense ($R_3$). In contrast, an LE learned by BSRL is:

$$\texttt{communication}(s) \leftarrow \texttt{Contains}(s,a) \wedge a.\text{mood} = \texttt{imperative} \wedge a.\text{tense} = \texttt{present participle}$$

which is interpretable but may not make sense given domain semantics since an action satisfying these conditions may imply a label besides C. Figure 10 shows more LEs learned by RuleNN for labels T&T (term and termination) and L (liability).

### 5.3 Human-Machine Co-creation: User Study

Having shown that RuleNN learns explainable, high-quality LEs, we were interested in finding out whether domain experts find the same and in particular, whether the interaction improves the LEs? 4 data scientists, with knowledge of NLU and FOL, were given 188 LEs learned for C. The goal was to select LEs whose semantics could be verified. Via the GUI mentioned earlier, participants could modify LEs (by dropping/adding predicates) and evaluate them on Contracts training set. Each participant took half an hour to select $\approx$ 6-8 LEs. This reduction from 188 LEs translates to a 96% model compression and shows that with human's expertise, RuleNN's LEs can be made smaller and thus more interpretable. To model collaborative and iterative development in the real-world, we union LEs produced by each subset of 3 participants to attain 4 explainable models. As Figure 8 (e) shows, 3 of these outperform BiLSTM by $\approx$ 25% in terms of F-measure (precision and recall's harmonic mean). As an example LE modification, consider $R_3$ (Figure 9) which contains the predicate $a.\text{tense} = \texttt{future}$. Since a sentence may imply communication even if it is not in future tense (e.g., Figure 1's sentence is in present tense), participants dropped this predicate to improve the LE's recall by 5% (precision remained $\approx$ 75%). Even if we learn the right patterns (many Contracts' sentences are in future tense), domain expertise may still aid generalization thus going beyond available training data. Yang et al. (2019) provides more details on the user study, design of the UI used to conduct it and related aspects.

[3]Fixing errors in SRL's output is out of scope of this work.

## 6 Conclusion and Future Work

Our experiments indicate that neuro-symbolic RuleNN outperforms other rule induction techniques in terms of efficiency and quality of rules learned even in the presence of challenging conditions such as class skew. Allowing domain experts to instill their expertise into LEs can also enable human-machine co-creation of explainable models.

Ideas presented here are general enough to enable other applications. RuleNN can be used for any MIL task assuming predicates are given and PGMs can be used to learn combinations of base predicates $\mathcal{P}$ even if the structure of the rule differs from LEs. As an extension, it may even be possible to determine the number of LEs $k$ from the data using recurrent neural networks (Yang et al., 2017). RuleNN can learn rules combining any previously built classifier's output probabilities (assuming such probabilities lie within $[0, 1]$). Note that, RuleNN's philosophy is distinct from other explainable AI approaches (Ribeiro et al., 2016; Serrano and Smith, 2019). We show that it is possible to learn human-interpretable models by designing neural networks keeping explainability in mind.

As opposed to learning an explainable model (e.g., RuleNN), one may also choose to *explain* a black-box model. Such efforts are usually restricted to explaining outcomes and only provide a shallow understanding of the overall model, if at all (Guidotti et al., 2018). While recent embeddings (Devlin et al., 2019) may lead to improved accuracy, these remain poorly understood (Moradshahi et al., 2019). One avenue of future work is to learn explainable rules that domain experts can interact with on top of such embeddings. Another is to learn rules and dictionaries jointly, which may also aid sentiment analysis (Wilson et al., 2005).

## Acknowledgments

We would like to thank the reviewers for helpful feedback. We would also like to acknowledge the help of a lot of people not present in the author list which was instrumental in making this research possible: Shivakumar Vaithyanathan (for thought provoking discussions), Yiwei Yang, Walter S. Lasecki, Eser Kandogan (for help building the UI which made the user study possible), Diman Ghazi, Poornima Chozhiyath Raman, Ramiya Venkatachalam, Vinitha Yaski and Sneha Srinivasan (for help with the user study). This work was done while Siddhartha Brahma and Matthias Boehm were at IBM Research.

## References

Jaume Amores. 2013. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine bias. www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Luke Bjerring and Eibe Frank. 2011. Beyond trees: Adopting MITI to learn rules and ensemble classifiers for multi-instance data. In *International Conference on Advances in Artificial Intelligence*.

BlackBoxNLP. 2019. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Hendrik Blockeel, David Page, and Ashwin Srinivasan. 2005. Multi-instance tree learning. In *ICML*.

Lingyang Chu, Xia Hu, Juhua Hu, Lanjun Wang, and Jian Pei. 2018. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *KDD*.

Domenico Corapi, Alessandra Russo, and Emil Lupu. 2010. Inductive logic programming as abductive search. *LIPIcs-Leibniz International Proceedings in Informatics, Vol. 7. Schloss DagstuhlLeibniz-Zentrum fuer Informatik*.

Andrew Cropper and Stephen H. Muggleton. 2015. Logical minimisation of meta-rules within meta-interpretive learning. In *ILP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Thomas G. Dietterich, Richard H. Lathrop, and Tomas Lozano-Perez. 1997. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*.

Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. 2017. Logic tensor networks for semantic image interpretation. In *IJCAI*.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural logic machines. In *ICLR*.

Francesc Esteva and Lluis Godo. 2001. Monoidal t-norm based logic: Towards a logic for left-continuous t-norms. *Fuzzy Sets and Systems*.

Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data. *JAIR*.

Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*.

Lise Getoor and Ben Taskar. 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM Computing Surveys*.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *ICLR*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Dan Jurafsky and James H. Martin. 2014. *Speech and language processing*, volume 3. Prentice Hall, Pearson Education International.

Seyed Mehran Kazemi and David Poole. 2018. Relnn: A deep neural model for relational learning. In *AAAI*.

Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. 2011. Learning markov logic networks via functional gradient boosting. In *ICDM*.

Stanley Kok and Pedro Domingos. 2010. Learning markov logic networks using structural motifs. In *ICML*.

Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. 2008. SystemT: A system for declarative information extraction. *ACM SIGMOD Record*.

Legal Categories. https://cloud.ibm.com/docs/services/discovery?topic=discovery-contract_parsing#contract_categories.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING*.

Adam Liptak. 2017. Sent to prison by a software program's secret algorithms. www.nytimes.com/2017/05/01/us/politics/sent-to-prison-by-a-software-programs-secret-algorithms.html.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*.

Mehrad Moradshahi, Hamid Palangi, Monica S Lam, Paul Smolensky, and Jianfeng Gao. 2019. Hubert untangles bert to improve transfer across nlp tasks. *arXiv preprint arXiv:1910.12647*.

Stephen Muggleton. 1996. Learning from positive data. In *Worshop on ILP*.

Stephen Muggleton, Wang-Zhou Dai, Claude Sammut, Alireza Tamaddoni-Nezhad, Jing Wen, and Zhi-Hua Zhou. 2018. Meta-interpretive learning from noisy images. *Machine Learning*.

Stephen H. Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. 2008. Toplog: ILP using a logic program declarative bias. In *International Conference on Logic Programming*.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*.

Nikolaos Pappas and Andrei Popescu-Belis. 2014. Explaining the stars: Weighted multiple-instance learning for aspect-based sentiment analysis. In *EMNLP*.

Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global vectors for word representation. In *EMNLP*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you? explaining the predictions of any classifier. In *KDD*.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning*.

Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. In *NeurIPS*.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *ACL*.

Parag Singla and Pedro Domingos. 2006. Entity resolution with markov logic. In *ICDM*.

Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. 2018. Lifted relational neural networks: Efficient learning of latent relational structures. *JAIR*.

Akash Srivastava and Charles Sutton. 2017. Autoencoding variational inference for topic models. In *ICLR*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

Xinggang Wang, Yongluan Yan, Peng Tang, Xiang Bai, and Wenyu Liu. 2018. Revisiting multiple instance neural networks. *Pattern Recognition*.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *EMNLP*.

Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*.

Yiwei Yang, Eser Kandogan, Yunyao Li, Walter S. Lasecki, and Prithviraj Sen. 2019. HEIDL: Learning Linguistic Expressions with Deep Learning and Human-in-the-Loop. In *ACL*.