Introducing DRAIL: a Step Towards Declarative Deep Relational Learning

Xiao Zhang[*] **Maria Leonor Pacheco**[*] **Chang Li** and **Dan Goldwasser**
{zhang923, pachecog, li1873, dgoldwas}@cs.purdue.edu

Abstract

We introduce DRAIL, a new declarative framework for specifying Deep Relational Models. Our framework separates structural considerations, which express domain knowledge, from the learning architecture to simplify the process of building complex structural models.

We show the DRAIL formulation of two NLP tasks, Twitter Part-of-Speech tagging and Entity-Relation extraction. We compare the performance of different deep learning architectures for these structural learning tasks.

1 Introduction

Building statistical models capable of dealing with realistic problems require making predictions over multiple, often interdependent, variables. In such settings, correctly capturing the dependencies between these variables often takes precedence to the specific algorithm used for estimating the models' parameters. Capturing these dependencies relies on compiling expert knowledge about the problem domain into the statistical model, and in recent years several machine learning systems offering intuitive interfaces for defining the dependencies between predictions were suggested (Domingos et al., 2006; McCallum et al., 2009; Rizzolo and Roth, 2010; Bach et al., 2015; Kordjamshidi et al., 2015).

On the other hand, end-to-end deep learning methods, which are becoming increasingly popular, take an almost opposite approach. These methods map the complex input object to desired outputs directly, without decomposing the decision process into parts and modeling their dependencies. The recent advances in deep learning allow these methods to successfully learn such mappings over very high dimensional latent features space (Duchi et al., 2011; Srivastava et al., 2014; Bahdanau et al., 2014).

At first glance these two trends seem almost contradictory, as the first highlights the importance of an easy-to-define, interpretable models, and the second focuses on finding complex non-linear mapping from the raw inputs to outputs that are difficult to interpret, and its definition requires considerable technical expertise. However, as we argue in this paper, these two objectives are not at odds, but rather express the specific considerations required at different levels of abstraction. We suggest to break the dependency between the two layers, and present a framework for supporting it, by separating the definition of structural elements and their dependencies from the specific learning architecture used for learning them. In this paper we describe the steps we took towards building a *declarative* framework for **D**eep **R**elat**i**on**a**l **L**earning (DRAIL).

DRAIL is a declarative modeling language for defining structured prediction problems, that separates between the modeling layer, which defines the high level dependencies between variables, and the learning layer, which defines the learning architecture that will be used and its parameters. From a modeling perspective, DRAIL is very similar to other declarative languages such as Markov Logic Networks (MLN) (Domingos et al., 2006) and Probabilistic Soft Logic (PSL) (Bach et al., 2015), and uses first order logic as a template language for defining factor graph templates, which are instanti-

[*] Equal contribution.

ated from given data.

A DRAIL program is defined over a set of predicates, which can represent either observed values or output predictions. We can define complex dependencies by connecting these predicates using rules of the following form:

$\mathtt{O(x,c)} \wedge \mathtt{O(x,d)} \Rightarrow \mathtt{P(x,y)}$

$\mathtt{P(x,a)} \Rightarrow \neg\mathtt{P(x,b)}$

In this example, `a,b,c` are constant symbols, `x,y` are variables, $\mathtt{O}(\cdot)$ is a predicate capturing observed properties of the input, and $\mathtt{P}(\cdot)$ is a predicate, representing a predicted property. The first decision rule, captures the mapping between observed properties of an input object and a prediction, while the second rule captures the dependency between two predictions (specifically, it states one output assignment for an input object prevents the assignment of another). Each one of the rule templates is associated with a score, either learned from data, or determined by the user, and the overall decision is made by finding the optimally scored assignments of output values to variables, by performing MAP inference. This flexible framework allows DRAIL to include both soft constraints, which quantify the dependency between different output decisions, and hard constraints, which force these dependencies by manually assigning the rule the highest possible weight.

Once the structural dependencies between the elements of the model are determined using the decision rules, we can turn our attention to learning considerations. From that perspective each rule defines a factor graph template, and using data we learn a scoring function for it. We learn the parameters of the scoring function for each rule using a deep learning architecture, which can be different for each rule, and normalized into a probability distribution, to allow global inference over all competing values. This flexibility is the key difference between DRAIL and other declarative learning frameworks: as it makes no distinction between base classifiers and soft constraints, the scoring function for both can be learned using highly expressive models.

We experimented with two well-known natural language processing structured prediction tasks, Twitter Part-of-Speech tagging (Gimpel et al., 2011) and Entity-Relation extraction (Roth and Yih, 2007). In section 3 we show how to define these tasks as DRAIL instances, and associate them with learning architectures. In section 4 we explain our inference procedure, converting a DRAIL instance into an Integer Linear program. We explain our learning approach in section 5. In our current experiments we used both Multi-layer Perceptron networks (MLP) and Recurrent Neural networks (RNN). We report our results in section 6.

2 Related Work

The difficulty of building complex machine learning models over relational data has attracted considerable attention in the machine learning community, and several high level languages for specifying the structure of different graphical models have been suggested. For example, BLOG (Milch et al., 2005) and CHURCH (Goodman et al., 2012) were suggested for generative models, and MLN (Domingos et al., 2006), PSL (Bach et al., 2015), FACTORIE (McCallum et al., 2009), and CCM (Rizzolo and Roth, 2010; Kordjamshidi et al., 2015) were suggested for conditional models.

In this paper we look into combining such declarative frameworks with deep learning models. Combining deep learning with structured models was studied by several works, typically in the context of a specific task or a specific inference procedure. These include dependency parsing (Chen and Manning, 2014; Weiss et al., 2015), transition systems (Andor et al., 2016) named entity recognition and sequence labeling systems (Ma and Hovy, 2016; Lample et al., 2016), and models for combining deep learning and graphical models for vision tasks (Zheng et al., 2015; Chen et al., 2015).

3 DRAIL Modeling Language

The DRAIL modeling language provides a general way to define relational learning problems that are highly structured. A relational model is specified in DRAIL using a set of weighted first-order logic rule templates that describe predictions and express the dependencies and constraints of a specific domain. Each rule is composed of: (1) a template definition written in first order logic, (2) the neural network architecture that will be used to learn the parameters of its scoring function, and (3) the set of feature functions to be extracted. Rules are then compiled into factor graphs, combining both predictions and observed variables. To further illustrate these definitions, we begin our explanation with two concrete examples of NLP applications.

Example 1: Part-of-Speech (POS) Tagging This task aims to map each word in a given sentence to its corresponding POS category (e.g., noun, verb, adjective, etc.).

Figure 1 describes an example of a simple model definition for the POS tagging task. Words and their corresponding POS tags have sequential dependencies that can easily be expressed in a declarative way. Our model consists of two rules: the first rule (line 1) conditions the POS tag only on the current word (i.e., similar to an *emission* feature), and the second rule (line 5) extends the dependency to both the previous word in the sentence and its tag assignment (similar to a *transition*). In these rules, `x` is the current word, `y` is the previous word, `z` is the POS tag assignment of the previous word, and `k` is the POS tag of the current word to be predicted.

```
1 rule:
2    def: Word(x) ⇒ HasPos(x,k)
3     network: MultiLayer, MultiClass
4     features: ["extract_twitter_glove"]
5 rule:
6     def: Word(x) ∧ HasPrevWord(x,y) ∧
          HasPos(y,z) ⇒ HasPos(y,k)
7     network: MultiLayer, MultiClass
8     features: ["extract_twitter_glove", "
          extract_tag"]
```

Figure 1: Modeling POS Tagging using DRAIL

In this example script, both rules are defined as multi-class prediction problems and are associated with Multi-layer neural network architectures (lines 3,7 respectively). We represent each word as a vector using Twitter Glove embedding (Pennington et al., 2014). We also use a vector representation for the previous POS tag (lines 4,8 respectively).

Example 2: Entity-Relation Extraction Our second example focuses on a simplified version of the relation extraction task (Roth and Yih, 2007; Kordjamshidi et al., 2015), which identifies named entities and their categories (PER, LOC, ORG) and two types of relations (LIVEIN, WORKFOR) over these entities. In Figure 2 we illustrate the model definition for this task, and write the structural dependencies of the problem using DRAIL. The model consists of three rules: the first rule (line 1) is used for predicting the entity category of a given phrase, and the second and third rules (lines 5 and 9, respectively) describe the two possible relations between pairs of phrases. We force the consistency between the entity and relation prediction types by encoding this knowledge as hard constraints (lines 13-16).

```
1  rule:
2      def: Phrase(x) ⇒ IsEntity(x,y)
3      network: MultiLayer, MultiClass
4      features: ["extract_entity_feats"]
5  rule :
6      def: Phrase(x) ∧ Phrase(y) ∧
           InSentence(x,z) ∧ InSentence(y,z) ⇒
           LiveIn(x,y)
7      network: MultiLayer, Binary
8      features: ["live_in_feats"]
9  rule :
10     def: Phrase(x) ∧ Phrase(y) ∧
           InSentence(x,z) ∧ InSentence(y,z) ⇒
           WorkFor(x,y)
11     network: MultiLayer, Binary
12     features: ["work_for_feats"]
13 const: LiveIn(x,y) ⇒ Entity(x,"Per")
14 const: LiveIn(x,y) ⇒ Entity(y,"Loc")
15 const: WorkFor(y,z) ⇒ Entity(y,"Per")
16 const: WorkFor(y,z) ⇒ Entity(z,"Org")
```

Figure 2: Modeling the Relation Extraction problem using DRAIL

DRAIL Elements The elementary units of the model are predicates, which represent relations in the domains. These can be binary relations between two variables (e.g., `HasPos(x,z)` where a word `x` has a POS tag `z`), or unary relations (e.g., `Phrase(x)`). The predicates can correspond to hidden or observed variables. In the latter case, the data corresponding to each predicate is loaded from raw files into a relational database that can later be automatically queried to instantiate groundings for each rule template. Otherwise, the assignment of predicted values is determined in an inference procedure afterwards.

Each rule template defines a learning problem, which is used to score different assignments to the head of the rule. Rules have the form A ⇒ B, where A (*body*) constitutes a conjunction of observations and predicted values and B (*head*) is the information to be predicted. We allow each rule to be defined as either a multi-class, multi-label or a binary learning problem.

The overhead of DRAIL is considerably light, considering the size of the tested data sets. We implemented a compiler to translate formatted rules provided by the user to rule templates speedily. DRAIL then creates a simple in-memory relational database instance by loading raw data, based on the rule templates created. In order to construct the training, validation and testing data sets, DRAIL queries the database to create inputs for models corresponding to designated rule templates. The overhead of DRAIL majorly lies on the database queries process, which can be alleviated by using a more sophisticated database that handles queries efficiently for larger volumes of data. As far as we know, a variety of matured industrialized database systems carry this merit. After the data sets are created, the training procedure will be the same as usual. Hence, we intend to improve this aspect in later formal releases.

We use several neural network architectures to learn a probability distribution over the different output value predictions. Our main idea is to have a model definition that is agnostic of the network architecture used. In this way, the type of network (e.g., MLP, CNN or RNN) as well as other hyper-parameters (e.g., the number of layers, the number of hidden units, the learning rate, etc.) can be easily tuned without changing the dependencies between output variables.

To be able to learn from observations, each observed grounding must generate features to feed into the associated neural network. Currently, we provide a basic feature extractor interface for users to extend. Features are programmable in the Python language and there is no limit as to which types of features can be included for a rule. We also provide a set of out-of-the-box features that can be directly used, such as word embedding and one-hot vector representations for predicate arguments. Since features are added programmatically, external resources can be easily incorporated.

Finally, `const` rules define hard constraints over the general problem. These constraints allow the user to inject domain or common-sense knowledge into the prediction problem. For example, in the relation extraction task, the LIVEIN relation can only be predicted between an entity phrase of type PERSON and an entity phrase of type LOCATION. These constraints do not require any learning, and they can be directly translated into inference constraints.

4 Inference

Given a specific instance, we assign values to the output variables by running an inference procedure, formulated as an Integer Linear Programming (ILP) problem over rule groundings.

A rule grounding is an instantiation of a rule template. We generate rule grounding by enumerating all possible values for the rule's variables given its domain. For example, the rule template `Word(x)` $\Rightarrow$ `HasPos(x,y)`, will be instantiated with each possible part-of-speech tag for each observed word.

We score the rule groundings by associating each template with a neural net. We denote the score of each rule grounding as w_i, the weight associated with rule grounding i. These weights are used as coefficients of the corresponding ILP variables in the objective function when performing global inference.

We introduce rule variables r_i for each rule grounding i and head variables h_j for each different head predicate j (and its negation $(\bar{h}_j)$) to indicate the activation of the variable. The objective function can then be expressed as

$$\arg\max_{\forall r_i} \sum_i w_i \cdot r_i$$

.

Note that in the objective function, we do not assign any weights to head variables as their values are entirely determined by constraints that ensure consistency.

In order to enforce consistency between variable assignments and dependencies among them, the following five types of constraints are taken into consideration in an ILP formulation.

negation constraints The first type constraints ensure exclusive activation of a head predicate and its negation at the same time. For example, $h_{\texttt{HasPos(a,b)}} + \bar{h}_{\texttt{HasPos(a,b)}} = 1$.

implied constraints Each rule template defines the dependency between body and head. This dependency is reflected between the rule groundings variable and the head variables in the body. For example, in the rule grounding `Word(a)` $\wedge$ `HasPrevWord(a,b)` $\wedge$ `HasPos(b,p)`

$\Rightarrow$ `HasPos(a,q)`, where a, b are words and p, q are part-of-speech tags, the constraint $r_{rule} \leq h_{\texttt{HasPos(b,p)}}$ is needed, as the whole rule is true only when the body is activated.

rule/head constraints One head predicate can be associated with multiple rule grounding variables. Let $r_i, i \in ruleset(j)$ denote the rule variables associated with the same head variable h_j, where $ruleset(j)$ is the set of rule groundings that share the same head predicate j. Activation of any rules in $ruleset(j)$ ensures the activation of the head variable, i.e. $h_j \geq r_i, \forall i \in ruleset(j)$. On the other hand, the activation of the head variable ensures the activation of at least one of its corresponding rule variables, i.e. $h_j \leq \sum_i r_i$.

binary/multi-class/multi-label constraints In many problems, we are facing multi-class or multi-label decisions. DRaiL guarantees this by adding suitable constraints. For instance, in the multi-class case, among all head variables h_j ($j \in decision(d)$) on the same entity, only one of them is activated while the others remain inactive, as a decision is made on which class to choose, i.e. $\sum_j h_j = 1$. Note that the constraints for binary predicates can be covered by the negation constraints mentioned above.

hard constraints from rule definitions Users can define hard constraints in the rule templates, which usually infuse prior knowledge and thus improve the prediction capacity. Rule groundings of these templates are dealt differently as the activation of such a rule depends on the activation of all body predicates. As an example, a hard constraint for entity-relation extraction problem discussed in this paper is `LiveIn(x,y)` $\Rightarrow$ `Entity(x, ``Per'')`.

We used the Gurobi Optimizer (Gurobi Optimization, 2015) to implement the inference module. As in many real world problem settings, the optimization problem based on the ILP formulation is computationally intractable, hence in practice we relax the inference procedure to linear programming (LP) problem by adapting the variable type from binary to continuous, within the range $[0, 1]$.

5 Learning

Each rule template in a DRaiL model file defines a learning problem, for scoring the mapping of the variables defined in rule body to its head. We designed the rules to include flexible definitions for the representation and architecture used for learning this scoring function.

In the training stage, rule groundings are unfolded from the data, using rule templates. For each rule template, the neural networks map the LHS (left hand side) of a rule to the RHS (right hand side) with some probability, given all possible groundings for the right hand side. This mapping is learned using a deep learning model, which uses a logistic regression on top and the output probability distribution will be used as scores over the multi-class classification problem in an succeeding inference procedure.

One of the main advantages of DRaiL is that the neural network architecture is separate from the structural model. For each deep neural network model, all the hyper-parameters (e.g., the type of neural network, the learning rate, etc.) can be configured and optimized. Consequently we can associate different deep neural network architectures with different rule templates, as suits the sub-problems best. This design not only grants more flexibility when dealing with a structural learning problem, but also enables users to experiment with more feasible choices. For example, in the entity relation classification problem, we can create a LSTM (long-short-term-memory) model for entity tagging and then a Multi-layer Perceptron model for relation classification. In our experiments we used two different architectures, a Multi-layer Perceptron model and a Recurrent neural network, which are briefly described in the following sections.

5.1 MLP (Multi-layer Perceptron)

MLP is a simple yet widely used feed-forward artificial neural network model mapping input data onto a set of appropriate outputs. It contains several layers of nodes as a directed graph. Nodes in each layer are fully connected and an activation function is applied to each node except the input nodes. MLP has been proved to be a useful modeling tool, capable of approximating any function (Cybenko, 1989), hence it can be directly applied to data that are linearly inseparable. In our experiments we used a simple three-layer MLP that can be formulated as follows:

$$h^s = sigm(\boldsymbol{W}[x_1; x_2; p^s]),$$
$$y^s = softmax(h^s),$$

where the ; operator means concatenation of input vectors, and p^s is the feature representation of a hidden variable s. Visually, this model can be illustrated by Figure 3.

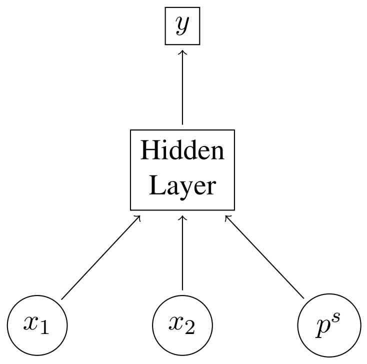

Figure 3: Multi-layer Perceptron Model for DRAIL.

5.2 RNN (Recurrent Neural Networks)

It has been repeatedly demonstrated that Recurrent Neural Networks are a good fit for sequence labeling tasks (Elman, 1990; Graves, 2013). We therefore used it for tackling the POS tagging task. Since our model (see section 3) for the POS task captures the dependency between the previous prediction and the current one, we formulate the RNN to accommodate this prediction task by including the unobserved predicates on the LHS of the rule templates as hidden variables. In order to break the dependency of the hidden predicates and the recurrent information in the model, we concatenated feature representations of the hidden variable with the recurrent representations that entail the history information and feed them to the final softmax layer to yield scores. This can be defined by the following equations:

$$\begin{aligned} h_t &= [sigm(\boldsymbol{W_x} x_t); sigm(\boldsymbol{W_h} h_{t-1})], \\ h_t^s &= [h_t; p_t^s], \\ y_t^s &= softmax(h_t^s) \end{aligned} \tag{1}$$

where p_t^s is the feature representation of hidden variable s at step t. A graphical demonstration of this model is shown in Figure 4.

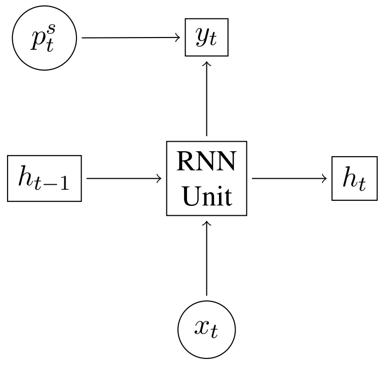

Figure 4: Recurrent Neural Network Model for DRAIL.

Possible future extension of deep neural network models include but are not limited to: LSTM (Long short term memory) model, GRU (Gated Recurrent Unit) model, Recursive Neural Network and Attention models. In addition, though our current learning implementation is training local learning models in the training stage combined with global inference in the testing stage, which is analogous to an MEMM model, in future work we plan to extend DRAIL to global learning in the training stage, similar to Conditional Random Fields (CRF).

6 Experiments

To demonstrate the generalization ability of our system to a variety of NLP tasks, we evaluated DRAIL on two structured prediction problems: Twitter Part of Speech (POS) tagging task (Gimpel et al., 2011) and the Entity-mention-Relation extraction task (Roth and Yih, 2007).

6.1 Part of Speech Tagging for Twitter

To tackle the Twitter POS tagging problem, Gimpel et al. (2011) used a CRF and defined a comprehensive set of features specific to the Twitter domain. We modeled the POS tagging problem in DRAIL as described on section 3 and tested it on the same data. This data set contains $1,827$ tweets ($26,436$ tokens) in total, divided in three folds: train, validation and test and it encompasses 25 different tags.

We used different features in our experiments, including pre-trained word embeddings[1] (Mikolov et

[1] We refer to Mikolov et al (2013) as *W2V*, and to Pennington

al., 2013; Pennington et al., 2014) to take advantage of our deep learning system as well as the base features defined by (Gimpel et al., 2011): a feature type for each word, suffixes of size 1 to 3, capitalization patterns, and features to indicate whether the word contains digits and hyphens.

We built deep neural network models for each template separately and trained them locally using different configurations of the enumerated feature sets. The same set hyper-parameters were used across both models: a MLP model with one hidden layer of 100 hidden units. We used batched stochastic gradient descent (SGD) with batch size 200 to train the models. Additionally, to prevent over fitting, we applied an early-stop paradigm to determine the appropriate number of training epochs by using the validation set. In the prediction stage, Integer Linear Programming (ILP) was used to perform global inference on the test set and decide the POS tag for all words in a sentence. In order to speed up the inference, we relaxed ILP to Linear Program (LP), approximating the $\{0, 1\}$ variables as a float number within the range $[0, 1]$.

Results Experimental results can be observed in tables 1 and 2. Table 1 shows the advantage of using deep neural networks over a Maximum Entropy Markov Model (MEMM) and the CRF results reported by (Gimpel et al., 2011). We obtained an improvement of 4.51% and 2.31% respectively, even by training Neural Networks locally, with zero tuning effort and performing global inference only at prediction time. In addition, we defined two models, a *local* baseline, using only the emission features (i.e., without inference), and a skyline that used the *gold* previous POS tags (i.e., no inference is required to determine this information). DRAIL's results after global inference got quite close to using the gold dependencies in the same MLP model, missing the skyline by just 0.37%.

One of the main advantages of deep neural networks is the way they can exploit feature embedding. By looking at table 2 we can observe that word embeddings pre-trained on Twitter data help boost the performance of this task considerably. In contrast, the pre-trained word embeddings from Google news negatively impact the results, decreasing the performance drastically. We attribute this result to the linguistic style and language characteristics of tweets, which greatly differ from those of news articles. To confirm this hypothesis, we did a subsequent qualitative analysis and discovered that a great number of twitter tokens are not present in the Google news corpus.

These results provide evidence that training simple MLP models locally, using suitable pre-trained embeddings and applying global inference for prediction, can outperform shallow models with global training procedures, even without additional manual efforts to tune hype-parameters. Furthermore, we extended our deep neural networks model to Recurrent Neural networks in DRAIL, and integrated it with global inference. Due to time limitation, DRAIL is only tested with simple features, while still demonstrating comparable performance on this task.

Model	Feature set	Accuracy
CRF (Gimpel et al., 2011)	Base Features	83.38%
MEMM	Base Features	81.00%
DRAIL Local Prediction (Baseline)	Base Features + Twitter Embedding	84.20%
DRAIL Gold Dependencies (Skyline)	Base Features + Twitter Embedding	85.88%
DRAIL MLP with Global Inference	Base Features + Twitter Embedding	85.51%

Table 1: Comparison of DRAIL to other models

Model	Feature set	Accuracy
MLP with Global Inference	Google W2V	55.52%
	BOW	75.50%
	Twitter Embedding	78.35%
	Twitter Embedding + BOW	82.10%
	Twitter Embedding + Base Features	85.51%
Local RNN	BOW (randomized vector)	79.68%
RNN with Global Inference	BOW (randomized vector)	80.12%

Table 2: Accuracy of Twitter POS tagging using DRAIL with different architectures and feature sets

et al (2014) as *Twitter Embedding*

6.2 Entity-Relation Extraction

We used DRAIL to describe a joint model to extract named entities and relations between them. The data set used was created by (Roth and Yih, 2004). It contains 1441 sentences and 37261 phrases. There are four types of entities: People (1691), Location (1968), Organization (984), and Other (706) and five types of relations. Similar to previous work on this data set, we focused on two specific relations: LIVEIN (521) indicates a Person lives in a Location, and WORKFOR (401) indicates a Person works for an Organization. All other pairs of phrases do not hold any relation between them, which means the data set is highly skewed.

We used the model configuration outlined in Figure 2. For the entity classifier, a set of features were extracted from phrases with a window of size 4 around the target including itself. These features include words, word embedding, part-of-speech tags, suffix, prefix, gazetteers and capitalization patterns. For the relation classifier, we included the same set of features mentioned above for both phrases as well as a small list of indicator words additionally, like "live", "native", "employ", and its relative position to the two target phrases. We also used features from the path between two phrases on the dependency parsing tree. A deep neural network model was trained locally for each rule. These networks have one hidden layer and 100 hidden units (300 for relation classifiers). We used stochastic gradient decent with batch size 100 and AdaGrad (Duchi et al., 2011) to adapt the learning rate in training. Softmax probabilities from the neural networks were used as scores in each ILP instance to find the optimal solution.

Results The results using 5-fold cross-validation are shown in Table 3. We report the F1 score of the positive class, i.e. $F_{\beta=1}$. To show the modeling capacity and expressiveness of DRAIL, we also tested using 0-order MEMM for local models. For each configuration, we report the result obtained directly from locally trained classifiers, and the results after global inference. As it shows, the deep architecture has a higher impact on the relation extraction problem than on entities, demonstrating its efficacy on difficult decisions. Also, the global inference procedure helps improve the performance on relation extraction significantly. Even without incorporating numerous handcrafted features, the performance of DRAIL is commensurate to or better than previous work, including (Kordjamshidi et al., 2015) and (Roth and Yih, 2007). Note that results are not directly comparable because their data splits specifications were not available.

Model	PER	LOC	ORG	WorkFor	LiveIn
MEMM Local	93.96	88.80	78.82	54.12	43.56
MEMM Global	93.43	89.02	79.36	54.68	53.09
MLP Local	92.32	89.78	80.60	54.80	51.19
MLP Global	92.30	90.05	79.86	62.84	56.86

Table 3: The F1 of different local models with and without inference on Entity-Relation Extraction task using DRAIL, 5-fold cross validation

7 Discussion and Conclusion

This paper introduces DRAIL, an open-source[2] declarative framework for defining structural dependencies between probabilistic concepts trained using deep learning models. The experimental results on the Twitter POS tagging problem and the Entity-mention-Relation extraction task demonstrate the flexibility of our framework, which can be used for quick prototyping and evaluating the interplay between representation complexity and structural complexity. DRAIL takes advantage of both building dependencies between structures and mapping complex inputs to outputs, resulting in a richer and more flexible hypothesis class, and enriched feature representations at the same time. These merits enhance the prediction ability of DRAIL. Our current work looks into including efficient and effective joint training (CRF Neural Networks) into DRAIL, by taking into account assigning values to all the variables in a global model and back-propagating error to the whole structure simultaneously. A potential advantage is that we can pre-train the models locally, and use global training to boost the performance using the same set of parameters, by only changing the objective function. Our DRAIL framework carries the capacity to incorporate this training paradigm into itself without additional effort.

[2]we intend to release the code and data used in our experiments

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks.

Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2015. Hinge-loss markov random fields and probabilistic soft logic. arXiv:1505.04406 [cs.LG].

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*.

L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. 2015. Learning deep structured models. In *Proc. of the International Conference on Machine Learning (ICML)*.

G Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Pedro M Domingos, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. 2006. Unifying logical and statistical ai. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

J Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211, jun.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. pages 42–47.

Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. 2012. Church: a language for generative models.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

Inc. Gurobi Optimization. 2015. Gurobi optimizer reference manual.

Parisa Kordjamshidi, Dan Roth, and Hao Wu. 2015. Saul: Towards declarative learning based programming.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Andrew McCallum, Karl Schultz, and Sameer Singh. 2009. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013*.

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. 2005. Blog: probabilistic models with unknown objects. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

N. Rizzolo and D. Roth. 2010. Learning based java for rapid development of nlp systems. In *LREC*.

Dan Roth and Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In Hwee Tou Ng and Ellen Riloff, editors, *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 1–8, Boston, Massachusetts, USA, May 6 - May 7. Association for Computational Linguistics.

D. Roth and W. Yih. 2007. Global inference for entity and relation identification via a linear programming formulation.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proc. of the Annual Meeting of the Association Computational Linguistics (ACL)*.

Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr. 2015. Conditional random fields as recurrent neural networks. In *Proc. of the International Conference on Computer Vision (ICCV)*.