Hands-On Interactive Neuro-Symbolic NLP with DRaiL

Maria Leonor Pacheco[1]
Microsoft Research &
University of Colorado Boulder
maria.pacheco@colorado.edu

Shamik Roy
Purdue University
roy98@purdue.edu

Dan Goldwasser
Purdue University
dgoldwas@purdue.edu

Abstract

We recently introduced DRaiL, a declarative neuro-symbolic modeling framework designed to support a wide variety of NLP scenarios. In this demo, we enhance DRaiL with an easy to use Python interface equipped with methods to define, modify and augment models interactively, as well as with methods to debug and visualize the predictions made. We demonstrate this interface with two challenging NLP tasks: analyzing moral sentiment in political discourse, and analyzing opinions about the Covid-19 vaccine.

1 Introduction

Language in real world settings is complex and ambiguous, and relies on a shared understanding of the world for its interpretation. Most current NLP methods represent language in a latent high-dimensional space by learning word co-occurrence patterns from massive amounts of textual data (Devlin et al., 2019; Brown et al., 2020). This representation is very powerful, but it can be insufficient to capture non-linguistic context such as the physical, social and cultural environments (Bisk et al., 2020). Manually annotating these diverse sources of context is a major challenge, and as a result, interactive and humans-in-the-loop approaches are gaining popularity to enhance and correct NLP models (Lertvittayakumjorn and Toni, 2021). However, the complexity of the representation learned by deep learning models create challenges for the communication between humans and machines. To circumvent these challenges, most existing humans-in-the-loop techniques solicit people to provide feedback on individual predictions instead, or allow people to augment the dataset by providing additional examples (Wang et al., 2021). While straightforward, working in the space of the raw inputs does not take advantage of the ability of humans to make abstractions and reason over them, like forming concepts to generalize from observations to new examples (Rogers and McClelland, 2004), turning raw sensory inputs into high-level semantic knowledge (Navon, 1977), and deductively drawing inferences via conceptual rules and statements (Johnson, 1988).

Neuro-symbolic representations present us with an opportunity us to enrich expressive language representations. On the one hand, symbols can be used to represent higher-level concepts and abstractions to characterize the information expressed in the text without resorting to individual annotations. On the other hand, distributed representations can help us ground these concepts and generalize to linguistic variations. Moreover, symbolic rules allow us to explicitly model the dependencies between aspects of the language and higher-level abstractions and behaviors. Most importantly, neuro-symbolic representations are inherently explainable, making them particularly useful for interactive and humans-in-the-loop approaches. Recently, we introduced DRaiL (Pacheco and Goldwasser, 2021), a neuro-symbolic modeling framework for NLP. In this work, we enhance DRaiL with a Python interface to facilitate the interactive exploration of neuro-symbolic NLP models. We demonstrate how to model two challenging language scenarios using DRaiL, and propose a set of diagnostic and visualization operations to probe and debug DRaiL predictions. Then, we demonstrate how we can interactively enhance and modify DRaiL programs to correct mistakes and introduce additional knowledge. This work represents a first step towards an interactive neuro-symbolic framework. An executable version of this demo is publicly available, and it includes the full code flow to run an example[2]. The source code for DRaiL, as well as its documentation and additional examples have been

[1] Work done while the first author was at Purdue University.

[2] https://bit.ly/3uLH26s

Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 371 - 378

released to the community[3]

2 Case Studies

Morality Framing in Political Discourse We previously introduced Morality Framing, a knowledge representation framework for capturing sentence and entity level moral sentiment (Roy et al., 2021). It is built on top of the Moral Foundation Theory (MFT) (Haidt and Joseph, 2004; Haidt and Graham, 2007) that proposes six Moral Foundations (MFs). Morality Frames extend MFT by introducing entity sentiment dimensions. The MFs are considered as frame predicates, and positive and negative entity roles are associated with each predicate. For example, the MF 'Care/Harm' is motivated by entities that are either 'providing care' or 'doing harm' to a specific target entity.

Morality Frame Predicate: Care/Harm

[New cyber center]$_{CARING}$ will provide hands-on learning to prepare midshipmen to protect [US]$_{TARGET}$ from [cyber terrorists and thugs]$_{HARMING}$.

The full list of morality frames can be found in the original paper. Given a text, the task is to identify - (1) the predicate (MF), and (2) the moral roles of the entities mentioned in the text. The dataset contains 1.5k tweets by US congress-members annotated for MF and entity roles.. We also released a dataset of 9.5k unlabeled tweets on the abortion issue, which we explore in this demo.

The Covid-19 Vaccination Debate In previous work, we proposed a holistic analysis framework to analyze opinions about the Covid-19 vaccine (Pacheco et al., 2022). This framework builds on Morality Frames, and connects it with opinion analysis. In addition to predicting MFs and entity roles, we predict the stance with respect to the vaccine (i.e. pro-vax or anti-vax), and we model a set of repeating themes frequently used to discuss the vaccine in social media.

Stance: Anti-Vax, Theme: Government distrust

I never saw anything like this [government]$_{OPRESSING}$'s obsession with [citizens]$_{TARGET}$ getting the Covid vaccine. Is this a trial run for a socialist dictatorship?

Our analysis identifies the *stance* expressed in the post (`anti-vaccination`) and the *reason* for it (`distrust of government`). Given the ideologically polarized climate of social media discussion on this topic, we also aim to characterize the moral attitudes expressed in the text (`oppression`), and how different entities mentioned in it are perceived. The dataset contains 750 tweets geo-located in the U.S. annotated for morality frames and stance, as well as a set of themes identified interactively. We also released a dataset of 85k unlabeled tweets about the covid vaccine, which we explore in this demo.

[3] https://gitlab.com/purdueNlp/DRaiL

3 Problem Specification

In this section, we demonstrate how to model the scenarios described in Section 2. To model a problem in DRaiL, we need to decompose the domain into a set of entities, labels, predicates and probabilistic rules that express the different decisions and their inter-dependencies. To predict morality frames, we break down the problem into two main decisions: 1) the most prominent moral foundation expressed in the tweet, and 2) for each entity mentioned in the tweet, the role they playing. In addition to this, we include some contextualizing information. For political tweets, we model the topic being discussed, the author of the tweet, and their party affiliation. In the case of covid-19 debate, we model the stance (i.e. pro or anti vax), as well as the main theme highlighted in the argument (e.g. government distrust).

In this demo, we present a Python API that allows us to instantiate and learn DRaiL programs. The API is centered around a `Learner` class. We currently support two types of learners, a `LocalLearner` in which rule weights are learned independently of each other, and a `GlobalLearner` in which all rule weights are learned jointly. The learner receives a set of parameters, including the inference algorithm and loss function to be used, as well as the learning rate.

```
from drail.learn.global_learner import GlobalLearner

learner = GlobalLearner(
    infer_algorithm="ad3",
    loss_fn="hinge_loss",
    learning_rate=2e-5)
```

Entities and Predicates Entities are the base elements in a DRaiL program. Entities are named, and can correspond to either symbolic elements (e.g. a topic) or attributed elements (e.g. a tweet associated with its textual content). Then, we can specify relations between one or more entities in DRaiL. Relations can correspond to observed or predicted information. When observations are available for a particular relation, either as input information or as training data, it can be passed to DRaiL using column separated files. Each column in the file will

correspond to each of the entities involved in the relation. For example:

```
learner.define_entity("Tweet")
learner.define_entity("Entity")
learner.define_entity("Role")

learner.define_predicate("HasEntity",
    ents=["Tweet", "Entity"],
    data_file="has_entity.txt")

learner.define_predicate("HasRole",
    ents=["Tweet", "Entity", "Role"],
    data_file="has_role.txt")
```

Rules and Constraints In DRaiL, decisions and dependencies between different decisions can be modeled using probabilistic rules. We can express these rules using templates of the form: $P_0 \wedge P_1 ... \wedge P_{n-1} \Rightarrow P_n$, where the body of the rule template can contain observed or predicted predicates, and the head corresponds to the output to be predicted (given the body). Rules can be grounded in data, and each rule grounding is associated with a weight representing the likelihood of the rule grounding holding true. In DRaiL, these weights are learned using neural nets. For this reason, each rule template is associated to a feature function and a neural scoring function.

We support different types of rules. We can define simple rules that map observed inputs to predicted outputs, and estimate the likelihood of each possible assignment. For example, mapping tweets to MFs:

```
learner.define_rule(
    "IsTweet(T) => HasMf(T,M)^?",
    lmd=1.0,
    features=["tweet_bert"],
    nn=BertClassifier(config_r0))
```

Or we can write rules that capture the dependencies between different aspects, and estimate the likelihood them co-occurring. For example, MFs and vaccination stances:

```
learner.define_rule(
    "IsTweet(T) & HasStance(T,S)^? => HasMf(T,M)^?",
    lmd=1.0,
    features=["tweet_bert", "stace_1hot"],
    nn=Bert1HotClassifier(config_r2))
```

Here, `lmd` is a hyper-parameter that can be used to manually tune the importance of each rule. Given a learned weight `w` for a given rule, its final weight will be calculated by multiplying `lmd*w`. We use `?` after a predicate to signal that this is a predicate that is not observed, and should be predicted.

Finally, we can also write hard dependencies or constraints that enforce behaviors. For example, enforcing entities to maintain the same polarity when mentioned in tweets with the same stance:

```
learner.define_hardconstr(
    "HasEntity(T1,E) & HasEntity(T2,E) &
    ↪  HasStance(T1,'anti-vax')^? & HasStance(T2,
    ↪  'anti-vax')^? & HasSentiment(T1,E,'neg')^?  =>
    ↪  HasSentiment(T2,E,'neg')^?")
```

Feature Extractors and Scoring Functions DRaiL gives us the flexibility to define any feature function and neural architecture to represent rules and learn their weights. To define feature functions, we need to extend DRaiL's `FeatureExtractor` class. This programmatic interface gives us a lot of flexibility with passing and importing resources, as well as manipulating features:

```
from drail.features.feature_extractor import FeatureExtractor
from transformers import AutoTokenizer

class MF_ft(FeatureExtractor):
    def __init__(self, id2data):
        super(MF_ft, self).__init__()
        self.id2data = id2data
        self.tokenizer = AutoTokenizer.from_pretrained(
            'bert-base-uncased')

    def entity_bert(self, rule_gr):
        pred = rule_gr.get_body_predicate("HasEntity")
        (tweet, entity) = pred['arguments']
        text = self.id2data[tweet][entity]['text']
        bert_input = self.tokenizer.encode(text)
        return bert_input
```

The constructor allows us to pass any data structure. In the example above, we pass a dictionary that maps entity ids to their attributes (e.g. the text of the tweet). Then, we import the `transformers` library to obtain the inputs for BERT. Alternatively, this could be pre-computed and passed to the constructor directly. DRaiL allows us to obtain the predicates and arguments of each rule grounding with the function `RuleGrounding.get_body_predicate(name, position=0)`, which returns the predicate as a dictionary of the form `{"name": name, "arguments": [arg0, arg1, ...]}`. Custom `FeatureExtractors` can be instantiated in the `learner` by doing:

```
learner.fe = MF_ft(id2data="id2data.json")
```

DRaiL provides a similar programmatic interface to define neural scoring functions, which is built on top of PyTorch:

```
from drail.neuro.nn_model import NeuralNetworks
from transformers import AutoConfig, AutoModel, BertModel
import torch

class BertClassifier(NeuralNetworks):

    def __init__(self, config):
        super(BertClassifier, self).__init__(config)

    def build_architecture(self):
        self.bert_model = AutoModel.from_pretrained(
            'bert-base-uncased',
            add_pooling_layer=True)
        self.dropout = torch.nn.Dropout(config["dropout_prob"])
        self.hidden2label = torch.nn.Linear(config["h_dim"],
        ↪  config["o_dim"])
```

```
def forward(self, x):
    # will return feats that were defined in feat function
    bert_inputs = self.get_inputs(x)
    outputs = self.bert_model(bert_inputs['input_ids'],
    ↪  bert_inputs['attention_mask'],
    ↪  bert_inputs['token_type_ids'])
    pooled_output = outputs[1]
    pooled_output = self.dropout(pooled_output)
    logits = self.hidden2label(pooled_output)
```

Given that the neural architectures are defined programmatically, there is a lot of flexibility as to what each architecture can look like. For each rule, we can define a configuration dictionary `config` which can be passed to the constructor of the neural classifier. This allows us to specify variable parameters (e.g. number of hidden/output units), and to reuse classifiers for different rules.

To see the full set of rules that were tested for each use case, we refer the user to our previous work (Roy et al., 2021; Pacheco et al., 2022), and to the live demo and repository linked to this paper.

Grounding, Inference and Learning To instantiate our database and ground our rules, we need to call the `create_dataset` function and pass the directory that contains the files that were defined for each predicate. We are able to specify train, dev and test splits by creating filters in the database. This operation is useful when we want to perform K-fold cross-validation, as it allows us to dynamically change the splits in an execution loop.

```
db = learner.create_dataset("data_dir")
db.add_filter(
    name="isTrain", pred_name="IsTweet", entity_name="Tweet",
    ↪  ids=train_tweet_ids
)
```

DRaiL transforms all rule groundings into linear inequalities corresponding to their disjunctive form, and inference is then defined as an integer linear program:

$$\begin{aligned} \boldsymbol{y} \in \{0,1\}^n \, P(\boldsymbol{y}|\boldsymbol{x}) \equiv_{\boldsymbol{y} \in \{0,1\}^n} & \sum_{\psi_{r,t} \in \Psi} w_r \, \psi_r(\boldsymbol{x_r}, \boldsymbol{y_r}) \\ & s.t. \; c(\boldsymbol{x_c}, \boldsymbol{y_c}) \leq 0; \;\; \forall c \in C \end{aligned} \tag{1}$$

Where each rule grounding r, generated from template t, with input features $\boldsymbol{x_r}$ and predicted variables $\boldsymbol{y_r}$ defines the potential $\psi_r(\boldsymbol{x_r}, \boldsymbol{y_r})$, added to the linear program with a weight w_r. DRaiL implements both exact and approximate inference to solve the MAP problem, in the latter case, the AD^3 algorithm is used (Martins et al., 2015). Weights w_r are learned using neural networks defined over parameter set $\boldsymbol{\theta}$. For training using large-margin estimation, DRaiL uses the structured hinge loss:

$$\max_{\hat{\mathbf{y}} \in Y} (\Delta(\hat{\mathbf{y}}, \mathbf{y}) + \sum_{\psi_r \in \Psi} \Phi_t(\boldsymbol{x_r}, \hat{\boldsymbol{y}}_{\boldsymbol{r}}; \theta^t)) - \sum_{\psi_r \in \Psi} \Phi_t(\boldsymbol{x_r}, \boldsymbol{y_r}; \theta^t)$$

Where Φ_t represents the neural net associated with rule template t, and parameter set θ^t. Here, $\mathbf{y}$ corresponds to the gold assignments, and $\hat{\mathbf{y}}$ corresponds to the prediction resulting from the MAP inference defined in Eq. 1. Note that alternative estimations are also supported. More details can be found in the modeling paper (Pacheco and Goldwasser, 2021).

Our Python API wraps up all of this functionality in just two functions: `train` and `predict`. Additional parameters can be specified to select the optimizer, use loss augmented inference, or hot start the parameters by training rules locally first. We have found that hot starting parameters locally consistently improves performance across tasks. This finding is in line with previous work experimenting with deep structured prediction objectives (Han et al., 2019). The `predict` function returns two elements: `results` has the aggregated predictions in a data structure that can be directly used to evaluate performance using the `sklearn.metrics` library, while `preds` contains the resulting set of active predicates.

```
from sklearn.metrics import classification_report

learner.train(db,
    train_filter="isTrain",
    dev_filter="isDev",
    patience=10,
    local_hot_start=True)

results, preds = learner.predict(db,
    test_filter="isTest")

y_gold = results.metrics["HasMF"]['gold_data']
y_pred = results.metrics["HasMF"]['pred_data']
classification_report(y_gold, y_pred, digits=4)
```

4 Interactive Evaluation and Debugging

In this section, we present an evaluation module equipped with functions to interactively debug and visualize DRaiL models. These functions are especially valuable when evaluating model predictions over unlabeled data, where we cannot directly measure performance. To showcase this capability, we use the sets of unlabeled tweets about abortion and the covid-19 vaccine described in Sec. 2.

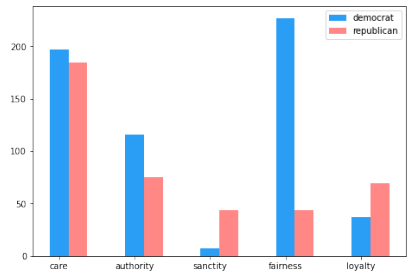
4.1 Visualizing and Interpreting Results

In this section, we present a non-exhaustive list of the visualization operations supported by our API. **freq_graph(pred,ent,filters)**: plots a bar graph of frequencies for entity `ent` in active `pred` predicates (e.g. `frequency_graph("HasMF", "MF")` will plot the distribution of MFs for all tweets. The optional parameter `filters` allows us to specify filters in the form of logical predicates. For exam-

ple, if we want to plot MF frequencies by political party, we can do:

```
learner.freq_graph("HasMF", "MF", filters=["IsAuthor(T,A) &
↪  HasParty(A, 'democrat')", "IsAuthor(T,A) & HasParty(A,
↪  'republican')"])
```

Note that more than one filter can be used to compare frequencies. If no filters are passed, general frequencies will be plotted (one bar per value).



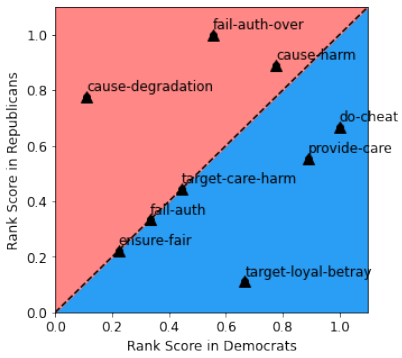
freq_ents(pred,ent,k,filter): outputs the top k most frequent ent entities in predicate pred.

```
learner.freq_ents("HasEntity", "Entity", k=10)
```

```
[('women', 325),
 ('abortion', 235),
 ('life', 207),
 ('wade', 115),
 ('trump', 111),
 ('roe', 104),
 ('planned parenthood', 91),
 ('reproductive rights', 90),
 ('health care', 62),
 ('unborn', 56)]
```

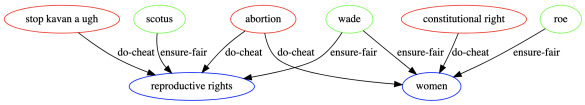
diag_rank_graph(pred,ent,ent_inst,top_filter, bottom_filter): plots a graph that visualizes the normalized rank scores based of the frequencies of entity ent=ent_inst in active predicates pred. This graph uses a diagonal to contrast the frequencies of the predicate activations that satisfy the top_filter and the bottom_filter. For example:

```
learner.diag_rank_graph("HasEntity", "Entity", "planned
↪  parenthood", top_filter="IsAuthor(T,A) &
↪  HasParty(T,'republican')", bottom_filter="IsAuthor(T,A) &
↪  HasParty(T,'democrat')")
```



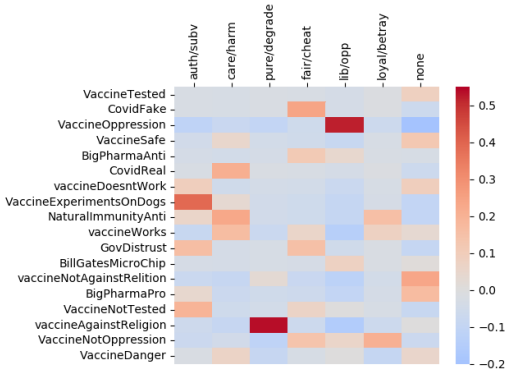
ent_rel_graph(pred,k,filter): plots an entity-relation graph for active predicates pred. Optionally, a k can be used to limit the graph to the top k most frequent activations. For example:

```
learner.ent_rel_rank_graph("HasRole", k=8,
↪  filter="IsAuthor(T,A) & HasParty(T,'democrat') &
↪  HasMF(T,'fairness')")
```



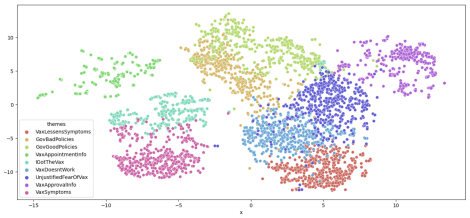
corr_matrix(pred_1,ent_1,filter_1,pred_2,ent_-1,filter_2): plots a correlation matrix between ent_1 and ent_1. Optionally, filters can be specified to constrain the examples considered.

```
learner.corr_matrix(pred_1="HasMF", ent_1="MF",
↪  pred_2="HasTheme", ent_2="Theme")
```



plot_embed(pred, ent, filter): plots a 2D visualization of the representation learned for a given entity. To reduce the dimensionality, we use t-sne (van der Maaten and Hinton, 2008). Note that DRaiL will learn a representation for each entity and relation in the program using the neural architecture specified. These representations can also be shared across rules. For more details, see (Pacheco and Goldwasser, 2021).

```
learner.plot_embed("HasTheme", "Theme")
```



4.2 Human Interventions

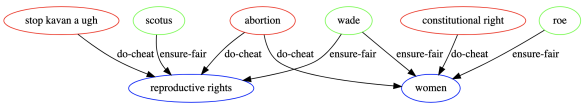
In this section, we focus on the ability of interactively correcting and enhancing DRaiL models. Going by the visualizations demonstrated above for political tweets, we can observe that overall, the results are what we would expect. However, we can spot some unexpected predictions. For example, in the normalized rank graph for *planned parenthood*, we found that the moral role *do-cheat* had a high *democrat* score, which contradicts the

general sentiment of liberals towards Planned Parenthood. Additionally, in the entity-relation graph, we observe the entity *abortion* being portrayed as a *cheating* entity in a high number of cases. To tackle this likely errors, we experiment with the following interventions:

Introducing bias with new rules and constraints Given that democrats generally have a pro-choice stance, we can introduce a rule that discourages a negative polarity for the *abortion* entity:

```
learner.define_rule(
    "IsTweet(T) & IsAuthor(A,T) & HasParty(A,'democrat') &
    ↪  HasEntity(T,'abortion') & HasPolarity(R,'neg') =>
    ↪  ~HasRole(T,'abortion',R)^?"),
    lmd=1.0,
    features=None,
    nn=None
```

To represent this rule, we set the features and neural net to None. This will make DRaiL learn a single weight for the rule, instead of learning a neural scoring function over a feature representation. Note that this is a design choice, and we always have the option of defining features and a neural classifiers for newly introduced rules. By adding this rule, we are able to alter our entity-relation graph:



Augmenting programs with new predicates While the rule introduced above altered our high-frequency entity-relation graph, upon closer inspection of the cases that were not covered by the soft constraint (using the **freq_ents** function to look at example tweets), we find that we likely still have errors. Tweets that have an overall negative tone are wrongly identified as portraying abortion in negative light. Some examples are: *we do not want to go back to the days before women had a constitutional right to abortion*, *president trump has said that women should face punishment for exercising their constitutional right to abortion*.

By looking at these examples, we can see that they talk about abortion as a *constitutional right*. To deal with this challenge, we can take advantage of the fact that we can represent entities in DRaiL using distributed representations, and introduce a new predicate that captures the similarity between a custom phrase explaining the concept of *constitutional rights* and the text of a tweet:

```
learner.define_entity("Phrase")
learner.define_latent_predicate(
    "MentionsConcept",
    ents=["Phrase", "Tweet"])
```

Given that we do not have supervision for this predicate, DRaiL allows us to define it as latent. Then, we can define two additional rules, the first one uses SBERT, a pre-trained sentence similarity model (Reimers and Gurevych, 2019) to capture the likelihood that a tweet mentions *constitutional rights*. The second one is a constraint that enforces tweets that frame abortion as a constitutional right to have the entity *abortion* in a positive role.

```
learner.define_rule(
    "InEvent(T,Z) => MentionsConcept('abortion is a
    ↪  constitutional right', T)^?",
    lmd=1.0,
    features=["tweet_bert"],
    nn=SBERT())
learner.define_hardconstr(
    "InEvent(T,Z) & HasEntity(T, 'abortion') &
    ↪  HasPolarity(R,'neg') MentionsConcept('abortion is a
    ↪  constitutional right', T)^? =>
    ↪  ~HasRole(T,'abortion',R)^?"))
```

Upon further inspection of example tweets, we found that the addition of the new predicate reduced more than 50% of the remaining errors.

5 DRaiL vs. Other Systems

In previous work, we have positioned the modeling approach of DRaiL with respect to related work, including declarative languages to express probabilistic graphical models (Richardson and Domingos, 2006; Bach et al., 2017), relational and graph embeddings (Bordes et al., 2013; Schlichtkrull et al., 2018), and a comprehensive set of neuro-symbolic systems (Wang and Poon, 2018; Manhaeve et al., 2018; Cohen et al., 2020). While performing an exhaustive comparison between systems is beyond the scope of this demo, we refer the reader to the DRaiL modeling paper for this analysis (Pacheco and Goldwasser, 2021), as well as to the many successful applications of our modeling strategy (Pujari and Goldwasser, 2019; Jain et al., 2020; Widmoser et al., 2021; Lee et al., 2021; Roy et al., 2021; Mehta et al., 2022; Pacheco et al., 2022).

6 Summary

In this paper, we present an interactive API for DRaiL, a recently introduced neuro-symbolic modeling framework. We demonstrate how to use this API to model a challenging NLP problem, and interactively debug predictions on unlabeled datasets, where traditional evaluation techniques cannot be applied. We motivate the advantage of neuro-symbolic representations to communicate knowledge from humans to machines, and show that we can effectively enhance the performance of the model by interactively adding new knowledge.

References

Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. Hinge-loss Markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*.

Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. 2020. Experience grounds language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8718–8735, Online. Association for Computational Linguistics.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

William W. Cohen, Fan Yang, and Kathryn Mazaitis. 2020. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *J. Artif. Intell. Res.*, 67:285–325.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonathan Haidt and Jesse Graham. 2007. When morality opposes justice: Conservatives have moral intuitions that liberals may not recognize. *Social Justice Research*, 20(1):98–116.

Jonathan Haidt and Craig Joseph. 2004. Intuitive ethics: How innately prepared intuitions generate culturally variable virtues. *Daedalus*, 133(4):55–66.

Rujun Han, Qiang Ning, and Nanyun Peng. 2019. Joint event and temporal relation extraction with shared representations and structured prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 434–444, Hong Kong, China. Association for Computational Linguistics.

Ayush Jain, Maria Leonor Pacheco, Steven Lancette, Mahak Goindani, and Dan Goldwasser. 2020. Identifying collaborative conversations using latent discourse behaviors. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 74–78, 1st virtual meeting. Association for Computational Linguistics.

Ralph H. Johnson. 1988. Gilbert harman change in view: Principles of reasoning (cambridge, ma: Mit press 1986). pp. ix 147. *Canadian Journal of Philosophy*, 18(1):163–178.

I-Ta Lee, Maria Leonor Pacheco, and Dan Goldwasser. 2021. Modeling human mental states with an entity-based narrative graph. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4916–4926, Online. Association for Computational Linguistics.

Piyawat Lertvittayakumjorn and Francesca Toni. 2021. Explanation-based human debugging of NLP models: A survey. *Transactions of the Association for Computational Linguistics*, 9:1508–1528.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3753–3763, Red Hook, NY, USA. Curran Associates Inc.

André F. T. Martins, Mário A. T. Figueiredo, Pedro M. Q. Aguiar, Noah A. Smith, and Eric P. Xing. 2015. Ad3: Alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16(16):495–545.

Nikhil Mehta, Maria Pacheco, and Dan Goldwasser. 2022. Tackling fake news detection by continually improving social context representations using graph neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1363–1380, Dublin, Ireland. Association for Computational Linguistics.

David Navon. 1977. Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, 9(3):353 – 383.

Maria Pacheco, Tunazzina Islam, Monal Mahajan, Andrey Shor, Ming Yin, Lyle Ungar, and Dan Goldwasser. 2022. A holistic framework for analyzing the COVID-19 vaccine debate. In *Proceedings of*

the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 5821–5839, Seattle, United States. Association for Computational Linguistics.

Maria Leonor Pacheco and Dan Goldwasser. 2021. Modeling content and context with deep relational learning. *Transactions of the Association for Computational Linguistics*, 9:100–119.

Rajkumar Pujari and Dan Goldwasser. 2019. Using natural language relations between answer choices for machine comprehension. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4010–4015, Minneapolis, Minnesota. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Mach. Learn.*, 62(1–2):107–136.

T. Rogers and James L. McClelland. 2004. Semantic cognition: A parallel distributed processing approach.

Shamik Roy, Maria Leonor Pacheco, and Dan Goldwasser. 2021. Identifying morality frames in political tweets using relational learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9939–9958, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web*, pages 593–607, Cham. Springer International Publishing.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Hai Wang and Hoifung Poon. 2018. Deep probabilistic logic: A unifying framework for indirect supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1891–1902, Brussels, Belgium. Association for Computational Linguistics.

Zijie J. Wang, Dongjin Choi, Shenyu Xu, and Diyi Yang. 2021. Putting humans in the natural language processing loop: A survey. In *Proceedings of the First Workshop on Bridging Human–Computer Interaction and Natural Language Processing*, pages 47–52, Online. Association for Computational Linguistics.

Manuel Widmoser, Maria Leonor Pacheco, Jean Honorio, and Dan Goldwasser. 2021. Randomized deep structured prediction for discourse-level processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1174–1184, Online. Association for Computational Linguistics.