From Examples to Rules: Neural Guided Rule Synthesis for Information Extraction

Robert Vacareanu, Marco A. Valenzuela-Escárcega, George C. G. Barbosa, Rebecca Sharp, Mihai Surdeanu
University of Arizona
Tucson, AZ, USA
{rvacareanu, gcgbarbosa, msurdeanu}@email.arizona.edu
{marcovalenzuelaescarcega, bsharpataz}@gmail.com

Abstract

While deep learning approaches to information extraction have had many successes, they can be difficult to augment or maintain as needs shift. Rule-based methods, on the other hand, can be more easily modified. However, crafting rules requires expertise in linguistics and the domain of interest, making it infeasible for most users. Here we attempt to combine the advantages of these two directions while mitigating their drawbacks. We adapt recent advances from the adjacent field of program synthesis to information extraction, synthesizing rules from provided examples. We use a transformer-based architecture to guide an enumerative search, and show that this reduces the number of steps that need to be explored before a rule is found. Further, we show that *without training the synthesis algorithm on the specific domain,* our synthesized rules achieve state-of-the-art performance on the 1-shot scenario of a task that focuses on few-shot learning for relation classification, and competitive performance in the 5-shot scenario.

Keywords: rule-based information extraction, rule synthesis

1. Introduction

The "deep learning tsunami" that "hit" natural language processing (NLP) (Manning, 2015) has brought tremendous improvements in performance to most NLP applications. However, these benefits do not come for free. One drawback of deep learning is its opacity, which limits the ability for users to make incremental improvements to deployed systems. In particular, the entanglement of deep learning approaches means that "changing one thing changes everything" (Sculley et al., 2015).

In contrast, rule-based approaches are much more amenable to incremental improvements as each individual rule can be interpreted explicitly and unambiguously. This is critical for systems which will be deployed and maintained for long periods of time. Further, because rules encode expert knowledge, experts can write them without first curating a large number of examples.

However, an important drawback to rule-based approaches is that rule development is time consuming, and requires expertise in both the domain at hand and in linguistics. To mitigate this limitation, many directions before the "deep learning tsunami" focused on rule learning from examples (Yarowsky, 1995; Riloff, 1996; Collins and Singer, 1999; Abney, 2002; McIntosh, 2010; Gupta and Manning, 2014, *inter alia*).

Here we propose a novel method for rule synthesis from examples that combines the strengths of deep learning with the advantages of rule-based methods. By utilizing a self-supervised pre-trained transformer, we minimize the number of examples needed from the expert. By generating human-readable rules, the resulting grammar can be adjusted or extended incrementally as needs shift.

Our ability to generate rules from limited examples is key to our approach as this mimics a real-world setting, where the cost (both in terms of time and money) of annotating thousands of examples for training supervised approaches is a barrier for many. Accordingly, we evaluate our methods in a *few-shot* framework, to simulate a user providing a small number of examples from which to generalize.

The key contributions of this paper are:

(1) To our knowledge, we are the first to propose methods inspired from program synthesis for rule learning in IE. Our method includes a contextualized neural component that scores each intermediate state in the rule synthesis process based on its likelihood to lead to a good rule, with backtracking facilitated with the Branch and Bound algorithm (Land and Doig, 1960). All data and code needed to replicate are open-source and publicly available.[1]

(2) In an intrinsic evaluation, we show that our neural-guided rule synthesis reduces the number of search steps considerably compared to a synthesis method using static state scores. Importantly, similar to language models, our neural guiding function

[1]The code is available at this URL: `https://github.com/clulab/releases/tree/master/lrec2022-odinsynth`.

doesn't use in-domain training data, which means that it works without training or fine-tuning on any IE domain.

(3) In an extrinsic evaluation we demonstrate the validity of our rule synthesis approach. In particular, we evaluate on the Few-Shot variant of the TACRED relation extraction task (Zhang et al., 2017; Sabo et al., 2021), and show that our approach considerably outperforms the state-of-the-art BERT model on the harder 1-shot task by 3% F1 points.

2. Related Work

Our approach lies at the intersection of program synthesis and rule learning for IE.

Program synthesis: There has been a large body of work on methods for program synthesis, with a general focus on automatically generating code (Gulwani, 2011; Lee et al., 2016; Balog et al., 2016; Gulwani et al., 2017). In general, program synthesis requires a program space (the domain specific language), the user intent (specification), and a search algorithm, from which it produces an executable program. One popular method for program synthesis is *program by example* (Cypher and Halbert, 1993; Lieberman, 2001; Gulwani, 2012), which learns from specifications in the form of (input, output) pairs. These methods perform a deductive search over the program space until a successful program is produced. Our rule synthesis method is situated within this framework. Importantly, as the program space becomes larger, this deductive search is intractable without intervention. There are different forms of intervention, including heuristic pruning (Lee et al., 2016), and usage of a neural guiding function (Balog et al., 2016; Kalyan et al., 2018). In our work we make use of both, using a transformer network to guide our search as well as custom heuristics to prune whole branches from the search tree.

Beyond simply using neural networks to guide, there have been efforts to generate the final program using a neural sequence-to-sequence model, e.g. (Yin and Neubig, 2017). With these approaches, execution guidance is typically used to ensure that the generated program is valid in the Domain-Specific Language (DSL) (Wang et al., 2018). This is not needed in our approach, since we make predictions over the possible next states as determined by the DSL grammar, which ensuring that every generated program is valid.

Aside from program by example, other approaches use different forms of specifications. For example, (Dong and Lapata, 2018) and (Hwang et al., 2019) generate programs from a natural language description of the desired behavior. However, for our work, we choose to focus on examples, as we feel it is a more intuitive interface; it can be very difficult to describe explicitly the desired behavior of an IE rule, especially when the user does not have experience with linguistic structures.

Rule learning for information extraction: At a high-level, IE approaches fall in one of three camps: (a) methods that rely on rules or patterns (either manually crafted, learned, or extracted using a shortest path through the syntactic graph (Yarowsky, 1995; Riloff, 1996; Collins and Singer, 1999; Abney, 2002; McIntosh, 2010; Chiticariu et al., 2010; Gupta and Manning, 2014; Shlain et al., 2020), (b) approaches that use "traditional" machine learning (Mintz et al., 2009; Riedel et al., 2010; Hoffmann et al., 2011; Surdeanu et al., 2012), and (c) neural approaches (Zeng et al., 2015; Lin et al., 2016; Zhang et al., 2018; Guo et al., 2019). Our approach is closest to the first camp, in the sense that we output rules, but we use state-of-the-art methods from the last camp (such as transformer networks) to generate these rules.

From an application-centric point-of-view, our approach is similar with (Sa et al., 2016) and (Ratner et al., 2017), as it allows for a human-in-the-loop type of interaction. However, the main difference is that our proposed model does not require domain experts.

3. Problem Statement

In this effort, we address the problem of generating (or synthesizing) rules for IE from a few examples, satisfying two key constraints: we cannot assume experience with linguistics or machine learning from the users, and the approach must be domain-agnostic.

In our approach, the user is able to specify *what* they need extracted by highlighting portions of text in sentences they have selected. For example, if a user is interested in extracting *parent-child* relations, they might select a sentence such as *He was a son of David and Mary M Anderson*, and from that sentence they might highlight *He* and *Mary M Anderson* as the content of interest (see Figure 1).[2] Note that at no time in the process do they need to concern themselves with the underlying syntactic structure or language model. This information forms the input to our method, which then searches for a rule that matches *only* the highlighted part of the input sentence(s), such as the one shown in the figure.

Before we describe the actual algorithm, we introduce necessary terminology:

Specification: We call a (sentence, selection) pair, such as the one shown in Figure 1, a *specification.*

[2]Here we are focusing on binary relation extraction, so while *David* is also correct for this relation, it would be a separate specification sentence.

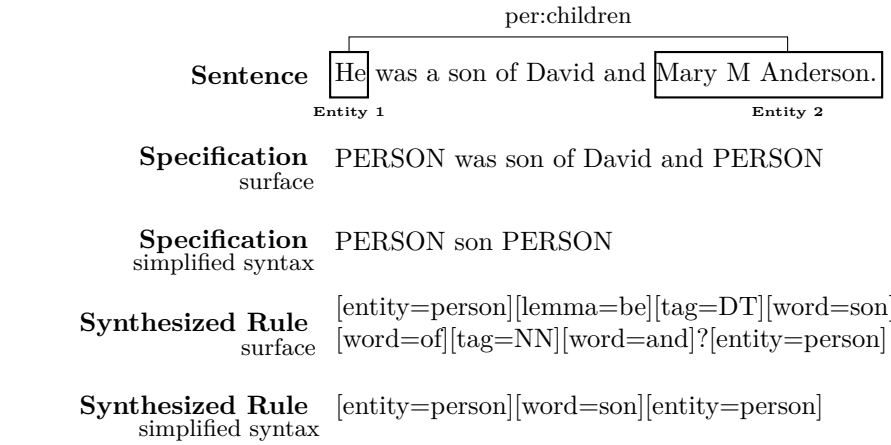

Figure 1: Example of input/output for our rule synthesis method. Our approach learns to generate individual rules that match a set of examples (or a *specification*), where each example is a highlighted span of text in a sentence. The "simplified syntax" lists the tokens contained on the shortest dependency path that connects the two entities. Note that this solution is not unique.

We overload the term specification to refer to one or multiple such pairs. Note that the selection may be empty, as in the case of counter-examples, in which case a generated rule should match nothing. The algorithm is required to generate a single rule that exactly matches all highlights in the provided specification, and nothing more.

Rules: We use Odinson (Valenzuela-Escárcega et al., 2020), a rule-based information extraction system, for rapidly evaluating a potential rule against the input. We chose Odinson because it brings two key advantages. First, Odinson rules are expressive; for example, a single rule can combine surface information with syntactic dependency paths. Second, the Odinson runtime engine is fast: its authors report that, due to its careful indexing of syntactic information, Odinson traverses syntax-based graphs six orders of magnitude faster than rule-based systems that operate without such an index.

As the search is informed by the syntax of Odinson, an important benefit of our approach is that every rule provided by the system is always a valid Odinson query. Though Odinson is able to support rules which combine surface and syntax, for this initial effort we focus only on rules that rely on token sequences, which usually come from surface information. We do, however, explore a linearization of syntactic paths, by using a sequence of tokens that consists of the tokens visited during a traversal of the shortest dependency path connecting the entities in the specification. For example, the shortest syntactic dependency path that connects *He* and *Mary M Anderson* in the sentence from Figure 1 contains the (unlabeled) dependencies: *son* → *He* and *son* → *Anderson*,[3] which is linearized to preserve sentence order as: *He son Anderson*. This representation is referred to as *simplified syntax* in Figure 1.

Placeholder and state: Each search begins with a *placeholder* (represented here as □), which can be replaced with any valid rule element during the rule synthesis process. We use the term *state* to refer to the information available at a given step of the rule generation; this information includes the current, intermediate form of the rule, as well as which parts of the specifications are matched. During rule generation, placeholders are iteratively expanded until we either (a) find a state that is a valid Odinson rule that satisfies the specification constraints, or (b) we reach a maximum number of steps, in which case no rule is produced. At each expansion, the algorithm determines the potential next states from the DSL, scores them based on their likelihood to be part of the completed rule, and adds them to a *priority queue* that is sorted in descending order of scores. The next state is then selected according to the queue. An example of possible expansion rules for □ is given here:

$\square \rightarrow \square\ \square$	(concatenation)
$\square \rightarrow [\square]$	(token constraint)
$\square \rightarrow \square \vert \square$	(alternation)
$\square \rightarrow \square\{?, *, +\}$	(quantification)

Note that the search space grows exponentially with depth, making a brute-force approach intractable. For example, attempting brute force to generate `[entity=person] [lemma=be] [tag=DT] [word=son] [word=of] [tag=NN]? [word=and]? [entity=person]`, one of the solutions to our previously introduced (sentence,

[3]We used CoreNLP (Manning et al., 2014) dependencies.

selection) input pair, yields ≫1 billion states to explore. Therefore, the efforts to both *prioritize* which states to explore first (Section 4.2) and to *prune* portions of the search tree that cannot yield a correct rule (Section 4.1.1) are crucial.

4. Method

In a nutshell, our proposed approach for rule generation uses enumerative search that is guided by a transformer-based scoring mechanism, and is optimized using search-space pruning heuristics. Our transformer model scores each potential next state, given the current state, such that the number of states to be explored is minimized. Specifically, our system consists of two main components:

A **searcher** (Section 4.1), with Branch and Bound (Land and Doig, 1960) as the underlying algorithm. The searcher uses the scores assigned by the scorer (below) to determine the order of exploration, choosing the state with the highest score, *regardless of its position in the search tree*. As such, it is important for the scorer to assign high scores to states that are in the sub-tree that leads to the desired final rule, and lower scores to all other states;

A **scorer** (Section 4.2), with a transformer backbone that is initialized with a pretrained model, but fine-tuned through self-supervision, i.e., over *automatically generated rules*. The scorer uses the current state and the specification to score each potential next state.

4.1. Enumerative Searcher

The searcher is responsible for exploring the states in priority order (as determined by the scorer), and deciding if a given state is successful (i.e., it is a valid query and correctly extracts the requested highlighted words and nothing more). The search space can be interpreted as a tree, where the root is the initial candidate solution and the children of a node n are the candidate solutions that the node n could expand to. Given this, the searcher can be seen as iteratively applying a sequence of three operations: (a) **Expand** the current state according to the DSL grammar,[4] (b) **Score** each expanded candidate next state and insert them into the priority queue, and (c) **Select** from the queue the state with the highest score to be the next state. We repeat this process until we find a solution or we reach our step limit.

Figure 2 shows a detailed example for the input sentence *He was a son of David and Mary M Anderson*, with *He and Mary M Anderson* as the highlighted span or selection. In this example, we generate one possible solution: `[entity=person] [word=son] [entity=person]`, which comes from the linearization of the dependency path between the two `person` named entities.

4.1.1. Pruning the search space

While the scorer determines the order of exploration, this is complemented by techniques for greatly pruning the search space to be considered. In particular, adapting the techniques of (Lee et al., 2016) to our use case, we prune states for which the *least restrictive* rule that could result from this state cannot completely match the highlighted specification, as nothing created from that subtree can be a solution. Consider the state: `[entity=person] [word=born]` □. The least restrictive rule resulting from this state would be one which matches the word *person*, followed by *born* and then 0 or more (unrestricted) tokens. If such a rule cannot completely match the highlighted tokens, then a valid solution cannot be found in that subtree so we prune the branch.

4.2. Scorer

The Scorer assigns a numerical value to states to establish the order of exploration. We explore two variants: as a baseline, we implement a **static** variant based on the components of a given state, and a **contextual** variant based on a self-supervised model that takes the current context into account.

4.2.1. Static weights

For this baseline, the score of a state is solely determined by its components. The cost of each state is constructed by summing the cost of its components with the cost of its node. For example, the cost of □ □ (concatenation) is: $cost(\square\ \square) = cost(\square) + cost(\square) + cost(concatenation)$

The costs for each operation were hand-tuned based on intuition (e.g., exploring negation takes a very long time as you need to consider everything some constrain *cannot* be, thus negation is given a higher cost), then optimized on a small external development set of sentences and specifications.

In addition to the hand-tuned nature of the static scorer, there are two main limitations. First, a given state will always receive the same score regardless of the sentence context or the previous state. Second, states with more components in their underlying pattern inherently have a higher cost because the score is summative. This is undesirable, as the states which expand to a solution should score higher than those which cannot, regardless of length.

4.2.2. Score augmentation based on estimation

To supplement the score from the static weights, we introduce an additional score that estimates how

[4] We always expand the leftmost placeholder first.

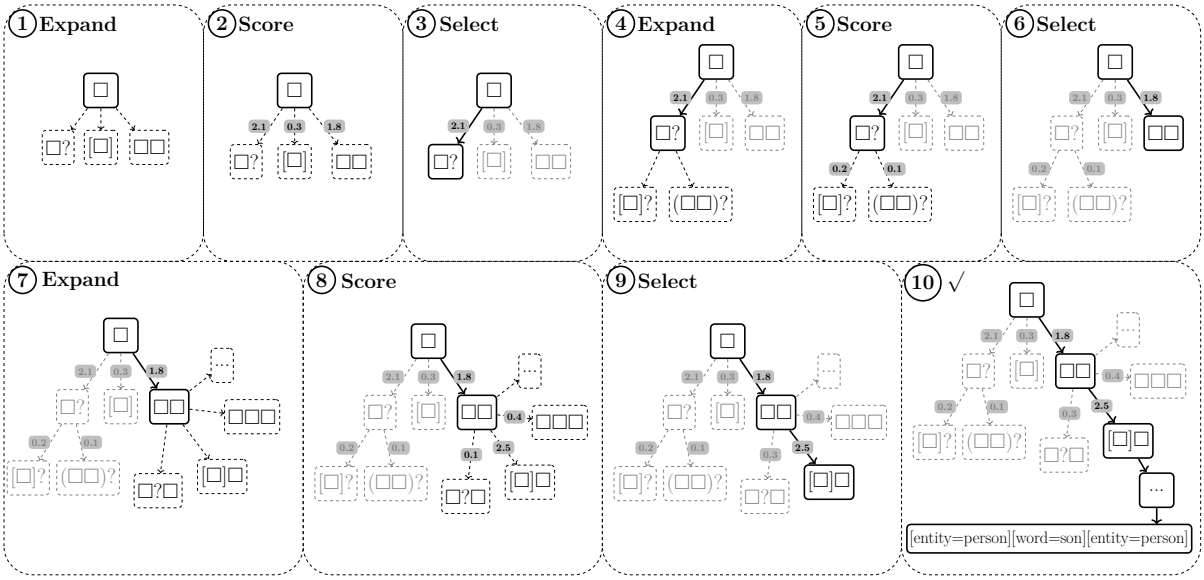

Figure 2: Conceptually, the algorithm consists of three main steps: **expand** the current state, **score** the possible next states, and **select** the state with the highest score. In the beginning, we start from the placeholder (□) and expand it (1). We score each expansion (2) and transition to the one with the highest score (3). We then expand and score again (4,5). At this point, notice that the state with the highest score is □ □, which is in a different subtree than the current state (□?). The enumerative searcher, with Branch & Bound as the underlying algorithm, transitions to the state with the highest score, regardless of the current position (6). This process repeats (7-9) until we arrive at a state that is a valid rule (10), or we reach a maximum number of steps and stop.

well a current (incomplete) rule matches the specification so far. For this, we remove all components of an incomplete rule that contain a placeholder and apply the remainder of the rule to the specification. We then boost the state's score for each specification token that is correctly matched, and penalize for each token incorrectly matched.
For example, for a rule such as `[entity=person] [tag=NN] [word=□]`, we remove incomplete components, resulting in the rule: `[entity=person] [tag=NN]`, which is matched against the specification. For each highlighted token matched, the function adds 1 to the score. For matches outside the highlight, the function adds −1. We observe that using this score augmentation favors more concrete constraints,[5] which help ground the rule to more lexical artifacts, but may hinder generalization.

4.2.3. Contextual weights

To address the limitations of the static weights, we propose a *contextual* scorer that utilizes the current context (i.e., the specification and the current state), to determine the cost of a candidate next state. Unlike our score augmentation, here we use the full specification, not just what is matched at a given time.
For this scorer, we use a transformer-based encoder to score each (current state, next potential state, specification) input. Intuitively, this score is the likelihood that the next potential state is better than the current state, which allows the scores to be comparable across all levels in the search tree.
Our contextualized scorer consists of a variant of BERT (Devlin et al., 2018; Turc et al., 2019)[6] with a linear layer on top. The BERT encoder input is a concatenation of: (1) linearized Abstract Syntax Tree (AST) of the current state (e.g., □), (2) linearized AST of the next potential state (□?), (3) and the (sentence, selection) specification. Since these concatenated components are fundamentally different, we differentiate between them by using different token type ids in the encoder. The tokens from the current state have a token type id of 1, and tokens of the next potential state have 2. We further differentiate between the highlighted and non-highlighted portions of the specification text in the same way, with token type ids 3 and 4, respectively.

4.2.4. Multiple sentences

So far we have used only a single sentence in our specification examples. Nevertheless, our system can handle multiple sentences and their highlights. We require the enumerative searcher to find a rule that would satisfy *all* the constraints for all sentences in the specification. When scoring, we score a (current state, next potential state, single-sentence specification) triple, and then average over all sen-

[5] More word constraints instead of tag or lemma constraints

[6] We experiment with multiple pre-trained variants of BERT, introduced in (Turc et al., 2019)

tences in the specification to obtain a final score for the (current state, next potential state) transition.

4.2.5. Training

Unlike the static scorer, the neural guiding function of the contextual scorer needs to be trained, which we do with self-supervision. Because there is no large corpus of Odinson rules, we artificially generate one with random spans of text that we randomly manipulate into rules. Our random-length text spans are chosen from the UMBC corpus (Han et al., 2013). Each token in this span is then randomly manipulated into an Odinson token constraint based on either word, lemma, or part-of-speech. For example, a span such as *the dog barked* might be converted to `[tag=DT] [word=dog] [lemma=bark]`. Then, to expose the model to additional rule components that encourage generalization (e.g., alternation, quantifiers), we add further manipulations, again with randomization. To add alternations, we build a temporary query by replacing one of the token constraints with a wildcard that can match *any* token and query the corpus for an additional sentence that has different content in that position. This new content is added as an alternation. For example, with the temporary version of the above query `[tag=DT] [word=dog] []`,[7] we might find *A dog runs*, resulting in the following alternation: `[tag=DT] [word=dog] ([lemma=bark]|[lemma=run])`. To add a quantifier (i.e., `*`, `+`, or `?`), we select a token to modify and a quantifier to add, and check the corpus to ensure that the addition of the quantifier yields additional results.

After generating each random rule, we build a corresponding specification by querying the UMBC corpus: the retrieved sentences and their matched spans constitute specification. However, having a specification and the corresponding rule is not enough to train our model. We also need a correct sequence of transitions from the initial placeholder to the final rule. For this, we use an Oracle to generate the shortest sequence of transitions, which we consider to be the correct sequence for our purposes. The Oracle consists of a simple procedure of generating the shortest path from the start symbol to the given rule, according to the grammar.[8] This sequence of transitions, together with the specification, forms the training data for our model. Note that we train *only* on this data, i.e., after this self-supervised training process the transformer's weights are fixed. We train using the cross-entropy loss and with a cyclical learning rate, as suggested by (Smith, 2017).[9] Further, we employ a curriculum learning approach (Bengio et al., 2009; Platanios et al., 2019), splitting the training data by sentence length and by pattern length. We did not tune our hyperparameters because we want to maintain the synthesis approach domain agnostic, rather than fine-tuning on a specific task. Our results indicate that this is the case.

5. Experiments and Results

We evaluate our system both intrinsically and extrinsically. The *intrinsic* evaluation is to determine whether or not the contextualized model reduces the number of search steps needed to find a valid rule. The *extrinsic* evaluation applies our rule synthesis approach to an information extraction task.

5.1. Intrinsic Evaluation

For the intrinsic evaluation, we compare how quickly a valid rule can be found when search is guided by our contextualized scorer versus the static scorers, measured on a held-out portion of our randomly generated dataset. We observe from Table 1 that the transformer-based contextualized approach finds *more* solutions in *fewer* steps. This demonstrates that the contextualized scorer is helpful for guiding the exploration of the rule search space.

5.2. Extrinsic Evaluation

To evaluate our approach extrinsically, we want to know how well it performs on an information extraction task. Ideally, we would evaluate our rule synthesis approach as it is intended to be deployed — with users providing specifications, and on large-scale information extraction projects. However, that is beyond the scope of the current, initial effort. A close proxy is few-shot relation extraction (RE), where a trained system receives a few (often 1 or 5) supporting sentences for a given relation and is then asked to recognize that relation in an unlabeled query sentence.[10] Recently, (Sabo et al., 2021) created a few-shot variant of TACRED (Zhang et al., 2017),[11] a RE dataset with 42 possible labels, including `no_relation`. Importantly, in this few-shot variant, the distribution of positives versus negatives was intentionally aligned with that of the real world.

On this task, we compare our rule synthesis approach with one strong baseline and several supervised approaches from previous work (i.e., tuned

[7] Note that the Odinson wildcard, `[]` looks similar to, but is not the same as our placeholder, □.

[8] Note that generating the transitions, once we know the rule is a simple problem. The problem of generating the rule (the task we tackle here) is the harder one.

[9] We train our scorer to predict 1 if the transition is correct and 0 otherwise. As such, we train it as a classifier and use it as a ranker during prediction.

[10] A notable difference between a human evaluation and the few-shot setup is that the former is interactive. That is, humans could refine the specification based on the results of the previous synthesis, but this is not straightforwardly done in the few-shot setting.

[11] `https://nlp.stanford.edu/projects/tacred/`

	Static Weights	Static + Score Aug.	Contextual Weights
Timeout	1k states	1k states	1k states
Rules found	283/1000	581/1000	862/1000
Ceiling avg	12.2	12.2	18.7
Ceiling median	14.0	14.0	17.0
Ceiling max	24	24	42
Ceiling min	9	9	9
Steps avg	432.3	98.1	55.9
Steps median	365.0	48.5	24.0
Steps max	999	676	845
Steps min	87	13	9

Table 1: Comparison of static and contextualized scorers in our rule synthesis approach on a held-out portion of our UMBC synthetic data. All methods were allowed to search until they reached the maximum number of explored states. Shown here are the number of rules successfully found, how many steps were required, and the ceiling (i.e., minimum steps possible using an Oracle). These statistics are averages over 1000 rules of different lengths.

model	5-way 1-shot	5-way 5-shot
Baseline	10.82 ± 0.01%	10.90 ± 0.01%
Sentence-Pair	10.19 ± 0.81%	-
Threshold	6.87 ± 0.48%	13.57 ± 0.46%
NAV	8.38 ± 0.80%	18.38 ± 2.01%
MNAV	12.39 ± 1.01%	**30.04 ± 1.92%**
Ours	**15.40 ± 1.21%**	24.16 ± 0.44%

Table 2: Results of our rule synthesis approach on the testing partition of Few Shot TACRED (micro F1 scores over target relations), compared with a baseline and previous supervised approaches.

on the disjoint background set). These results are provided in Table 2.

Baseline: At inference, each query sentence and each of the support sentences contain the *type* of the entities involved in the relation (e.g., `ORG`, `PER`, etc). Using this, we establish a smart random baseline. Specifically, the model randomly selects from relations whose supporting sentences have the same entity types, in the same order, as the query sentence, weighted by the number of supporting sentences with that relation. If there are none, we return `no_relation`. Additionally, for this baseline we make use of the disjoint background set. If there are sentences in that set with the same entity pair as the query, we choose one and add it to the supporting sentences with the label `no_relation`, available for random selection.

Previous supervised methods: We also compare our approach with the current, supervised state of the art (SOA) approaches:

Sentence-Pair (Gao et al., 2019): Concatenates the query sentence with each support sentence and runs the BERT sequence classification model over the concatenated text. If multiple support sentences are available per relation (e.g., in the 5-shot scenario), the score for a relation is obtained as the average over the scores for each sentence.

Threshold (Sabo et al., 2021): Assigns the `no_relation` class to query sentences if the similarity with the support sentences is below a threshold. Otherwise, it assigns the relation with the highest score, as in *Sentence-Pair*.

NAV (Sabo et al., 2021): A transformer-based relation classifier which uses the background training set to learn a vector for the `no_relation` class. At test time, the system computes the similarity between the query sentence and this learned vector, which represents the score for the `no_relation` class. The scores for the other relations are obtained using the BERT sequence classification model over the (query sentence, support sentence) pairs.

MNAV (Sabo et al., 2021): Conceptually similar to NAV, but learns multiple vectors for the `no_relation` class.

5.2.1. Extrinsic results

First, we note that our proposed baseline performs well, outperforming all of the more expensive BERT-based models on the harder 5-way 1-shot setting. Second, our proposed method surpasses the previous state-of-the-art method on the 5-way 1-shot setting, while obtaining competitive performance in the 5-way 5-shot setting. Besides the higher performance in the more challenging 5-shot 1-way

Span
ORGANIZATION , which is based in CITY
Synthesized surface rule:
`[entity=organization] [tag=","] [tag=WDT] [tag=VBZ] [tag=VBN] [tag=IN] [entity=city]`
Synthesized simplified syntax rule:
`[entity=organization] [tag=NN] [word=based] [entity=city]`
Span
ORGANIZATION , which represents ORGANIZATION
Synthesized surface rule:
`[entity=organization] [tag=","] [word=which] [tag=VBZ] [entity=organization]`
Synthesized simplified syntax rule:
`[entity=organization] [lemma=represent] [entity=organization]`

Table 3: Examples of our synthesized rules from the train partition of the Few-Shot TACRED. The relations for the three selected examples are: `org:city_of_headquarters`, `per:title`, and `org:subsidiaries`, respectively.

setting, note that the output of our approach is a set of *human-interpretable rules*, while the outputs of the other approaches are statistical models that produce the final label without explaining their decisions. In other words, previous work is much more opaque, and thus more difficult to interpret, debug, adjust, maintain, and protect from hidden biases present in the training data (Kurita et al., 2019; Sheng et al., 2019).

5.2.2. Synthesized rules

We give examples of specifications and the corresponding synthesized rule in Table 3. For space, we list only the highlighted spans. Notably, the system generalizes at different levels, depending on the data available. In longer surface rules our model prefers part-of-speech tag constraints, which helps generalization. In rules over simplified syntax, which are often shorter, our model tends to choose lemma or word constraints.

Overall, our results indicate that our approach provides a good compromise between the interpretability of hand-made rules and the performance of more opaque neural methods.

5.3. Implementation Details

We use experimented with multiple variants of BERT (Devlin et al., 2018), introduced in (Turc et al., 2019). The BERT encoder input is a concatenation of (1) linearized Abstract Syntax Tree (AST) of the current state, (2) linearized AST of the next potential state, and (3) the (sentence, selection) specification pair. We then use the embedding of the `[CLS]` token for prediction. We use a learning rate of $3e-5$. We used the `CyclicLR` learning rate scheduler (Smith, 2017). We used a curriculum style type of training (Bengio et al., 2009). We split our dataset into 3, based on the length of the sentence.

6. Conclusion

This paper is the first that proposed a synthesis algorithms for rule acquisition. Given the importance of explainability in deep learning, we feel this paper opens a new direction in information extraction (IE) that deserves more attention.

The proposed approach synthesizes rules from a user-provided specification. Given one or more (sentence, selection) pairs, the system is able to perform enumerative search to find a rule that successfully matches the requested information. Further, we demonstrated that we can utilize the context of the specification to improve the speed of the search, with self-supervised pretraining on an automatically generated, generic dataset. In an extrinsic evaluation that is modeled after the real-world scenario of a user needing to extract a novel relation with only a handful of annotations (i.e., few-shot relation extraction), we showed that our approach outperforms a state-of-the-art BERT model in the 1-shot scenario, and performs competitively in the 5-shot configuration.

Further, our approach is well-suited to a human-in-the-loop setting, where a user can select the desired information (or extraction) and request a pattern to be generated, without concern for the underlying representation. Given our approach, a user could request an alternative rule with the existing specification or augment it to clarify their needs. Importantly for many settings where systems are deployed long-term and information extraction needs may shift or expand, the resulting rules are able to be understood, modified, and maintained by human users.

Acknowledgements

This work was supported by NSF grant #2006583. Marco A. Valenzuela-Escárcega, Gus Hahn-Powell, and Mihai Surdeanu declare a financial interest in lum.ai. This interest has been properly disclosed to the University of Arizona Institutional Review Committee and is managed in accordance with its conflict of interest policies.

7. Bibliographical References

Abney, S. (2002). Bootstrapping. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 360–367.

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., and Tarlow, D. (2016). Deepcoder: Learning to write programs. *arXiv preprint arXiv:1611.01989*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F., and Vaithyanathan, S. (2010). Systemt: An algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 128–137.

Collins, M. and Singer, Y. (1999). Unsupervised models for named entity classification. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Cypher, A. and Halbert, D. C. (1993). *Watch what I do: programming by demonstration*. MIT press.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Dong, L. and Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742.

Gao, T., Han, X., Zhu, H., Liu, Z., Li, P., Sun, M., and Zhou, J. (2019). Fewrel 2.0: Towards more challenging few-shot relation classification. In *EMNLP/IJCNLP*.

Gulwani, S., Polozov, O., Singh, R., et al. (2017). Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119.

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330.

Gulwani, S. (2012). Synthesis from examples: Interaction models and algorithms. In *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 8–14.

Guo, Z., Zhang, Y., and Lu, W. (2019). Attention guided graph convolutional networks for relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 241–251.

Gupta, S. and Manning, C. D. (2014). Improved pattern learning for bootstrapped entity extraction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 98–108.

Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., and Weld, D. S. (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 541–550.

Hwang, W., Yim, J., Park, S., and Seo, M. (2019). A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Kalyan, A., Mohta, A., Polozov, O., Batra, D., Jain, P., and Gulwani, S. (2018). Neural-guided deductive search for real-time program synthesis from examples. *arXiv preprint arXiv:1804.01186*.

Kurita, K., Vyas, N., Pareek, A., Black, A. W., and Tsvetkov, Y. (2019). Measuring bias in contextualized word representations. *arXiv preprint arXiv:1906.07337*.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Lee, M., So, S., and Oh, H. (2016). Synthesizing regular expressions from examples for introductory automata assignments. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, pages 70–80.

Lieberman, H. (2001). *Your wish is my command: Programming by example*. Morgan Kaufmann.

Lin, Y., Shen, S., Liu, Z., Luan, H., and Sun, M. (2016). Neural relation extraction with selective attention over instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2124–2133, Berlin, Germany, August. Association for Computational Linguistics.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Linguistics*, 41(4):701–707.

McIntosh, T. (2010). Unsupervised discovery of negative categories in lexicon bootstrapping. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 356–365.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011.

Platanios, E. A., Stretcu, O., Neubig, G., Póczos, B., and Mitchell, T. M. (2019). Competence-based curriculum learning for neural machine translation. *CoRR*, abs/1903.09848.

Ratner, A. J., Bach, S. H., Ehrenberg, H. R., Fries, J. A., Wu, S., and Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, 11 3:269–282.

Riedel, S., Yao, L., and McCallum, A. (2010). Modeling relations and their mentions without labeled text. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer.

Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In *Proceedings of the*

national conference on artificial intelligence, pages 1044–1049.

Sa, C. D., Ratner, A. J., Ré, C., Shin, J., Wang, F., Wu, S., and Zhang, C. (2016). Deepdive: Declarative knowledge base construction. *SIGMOD record*, 45 1:60–67.

Sabo, O. M. S., Elazar, Y., Goldberg, Y., and Dagan, I. (2021). Revisiting few-shot relation classification: Evaluation data and classification schemes. *Transactions of the Association for Computational Linguistics*, 9:691–706.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28:2503–2511.

Sheng, E., Chang, K.-W., Natarajan, P., and Peng, N. (2019). The woman worked as a babysitter: On biases in language generation. 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 3398–3403.

Shlain, M., Taub-Tabib, H., Sadde, S., and Goldberg, Y. (2020). Syntactic search by example. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 17–23, Online, July. Association for Computational Linguistics.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472.

Surdeanu, M., Tibshirani, J., Nallapati, R., and Manning, C. D. (2012). Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 455–465.

Turc, I., Chang, M., Lee, K., and Toutanova, K. (2019). Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

Valenzuela-Escárcega, M. A., Hahn-Powell, G., and Bell, D. (2020). Odinson: A fast rule-based information extraction framework. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2183–2191.

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang, P.-S., Mao, Y., Polozov, O., and Singh, R. (2018). Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196.

Yin, P. and Neubig, G. (2017). A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Zeng, D., Liu, K., Chen, Y., and Zhao, J. (2015). Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1753–1762.

Zhang, Y., Qi, P., and Manning, C. D. (2018). Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215, Brussels, Belgium, October-November. Association for Computational Linguistics.

8. Language Resource References

Han, Lushan and L. Kashyap, Abhay and Finin, Tim and Mayfield, James and Weese, Jonathan. (2013). *UMBC_EBIQUITY-CORE: Semantic Textual Similarity Systems*. Association for Computational Linguistics.

O. Mahamane Sani Sabo and Yanai Elazar and Yoav Goldberg and Ido Dagan. (2021). *Revisiting Few-shot Relation Classification: Evaluation Data and Classification Schemes*.

Zhang, Yuhao and Zhong, Victor and Chen, Danqi and Angeli, Gabor and Manning, Christopher D. (2017). *Position-aware Attention and Supervised Data Improve Slot Filling*.