Completion Reasoning Emulation for the Description Logic $\mathcal{EL}^+$

Aaron Eberhart, Monireh Ebrahimi, Lu Zhou, Cogan Shimizu, and Pascal Hitzler
Data Semantics Lab
Kansas State University, USA
{aaroneberhart,monireh,luzhou,coganmshimizu,hitzler}@ksu.edu

Abstract

We present a new approach to integrating deep learning with knowledge-based systems that we believe shows promise. Our approach seeks to emulate reasoning structure, which can be inspected part-way through, rather than simply learning reasoner answers, which is typical in many of the black-box systems currently in use. We demonstrate that this idea is feasible by training a long short-term memory (LSTM) artificial neural network to learn $\mathcal{EL}^+$ reasoning patterns with two different data sets. We also show that this trained system is resistant to noise by corrupting a percentage of the test data and comparing the reasoner's and LSTM's predictions on corrupt data with correct answers.

Introduction

Machine learning and neural network techniques are often contrasted with logic-based systems as opposite in many regards. The challenge of integrating inductive learning strategies with deductive logic has no easy or obvious solution. Though there are doubtless many reasons for this, one major roadblock is that solutions and evaluations which work well for logic, or for deep learning and machine learning, do not work well in the opposite, and likely do not further the goal of integration. In this paper we present a new approach that embraces the liminality of this specific integration task. Our approach is by its very nature ill-suited to either machine learning or logic alone. But it tries to avoid the pitfalls of unintentionally favoring one paradigm over the other, aiming instead to grasp at something new in the space between.

Related Work

A popular approach for training neural networks to recognize logic is to use embedding techniques borrowed from the field of natural language processing (NLP). For instance, Makni and Hendler designed a system that layers Resource Description Framework (RDF) graphs into adjacency matrices and embedding them.(Makni and Hendler 2018) There is also considerable research into path-finding in embedded triple data using recurrent neural networks (RNNs) or LSTMs.(Xiong, Hoang, and Wang 2017; Das et al. 2016; 2017) Memory networks also successfully operate over triple embeddings and are capable of transferring training to work over triple embeddings from completely different vocabularies.(Ebrahimi et al. 2018) There is even an attempt to embed $\mathcal{EL}^{++}$ with TransE using a novel concept of n-balls, though it currently does not consider the RBox as well as certain $\mathcal{EL}^{++}$ axioms that do not translate into a triple format.(Kulmanov et al. 2019; Bordes et al. 2013)

Discussion

Many of the systems just mentioned are very successful, yet have a tendency to adopt familiar evaluation strategies from deep learning. Sure, it's great to be able to get a 99% F1-score. When we do not know the underlying structure of the data that we are using, this is often the best we can hope for. However, precision and recall do not make much sense in a knowledge base evaluation. For an integrated system, some new goals and evaluation strategies seem warranted that are neither completely deductive nor entirely empirical.

In a deductive reasoning system the semantics of the data is known *explicitly*. Why then would we want to blindly allow such a system to teach itself what is important when we already know what and how it should learn? Possibly we don't know the best ways to guide a complex network to the correct conclusions, but surely more, not less, transparency is needed for integrating logic and deep learning. Transparency often becomes difficult when we use extremely deep or complex networks that cannot be reduced to components. It also makes things difficult when we pre-process our data to help the system train by doing embeddings or other distance adjustments. When we embed all the similar things close together and all the dissimilar things far apart the system can appear to succeed, but we can't tell if it was the embedding or the system itself or one of a dozen other things that might have caused this.

In response to these and other concerns we have developed some research questions that we hope to address in this paper. Though we certainly will not be able to definitively answer all of them, we do hope that by starting this investigation we can demonstrate future potential for this type of inquiry.

Research Questions

1. Can a neural network emulate reasoner behavior?
2. Can we still achieve success without embeddings?
3. Can learning happen with arbitrary numerical names?
4. Will intermediate answers help our evaluation?
5. Is F1-score sufficient to evaluate an integrated logic and neural network system? If not, what would be?

Question one is the primary focus of this project, and the one to which all the others are subordinate. It is the intention of this paper and the prototype system we have implemented to demonstrate that this is a feasible goal. Whatever disagreements may arise concerning our later investigations, we will argue that this is a necessary and plausible alternative course of research.

The second question we have already mentioned in the last section. It is our intention to avoid any pre-processing that will allow the system to give correct answers without learning the reasoning patterns. In this way we can better verify whether we have truly solved the reasoning problem and not some other easier task.

The third question is related to the second, but imposes a more broad restriction on what we will be allowed to do. Not only do we want to avoid embedding the data in a way that will help the system to learn the answers. Also we must take care when we go from logic to numerical data, and back again, not to interfere with the latent semantics in the logic. By avoiding distance comparisons in the naming convention for the input we ensure that the names are arbitrary in both systems. The only distances we intend to measure are between predictions and answers.

The fourth question is very important for our conception of how future systems should work. In a logic-based system there is transparency at any level of reasoning. We cannot, of course, expect this in most neural networks. With a network that aims to emulate reasoner behavior, however, we can impose a degree of intermediate structure. This intermediate structure allows us to inspect a network part-way through and perform a sort of "debugging" since we know exactly what it should have learned at that point. This is a crucial break from current thinking that advocates more and deeper opaque hidden layers in networks that improve accuracy but detract from explainability. Inspection of intermediate answers could indicate whether a proposed architecture is actually learning what we intend, which should aid development of more correct systems.

The fifth and final question is a difficult but necessary question, likely beyond the scope of this paper. For now, it is primarily a reminder that the method we choose intentionally does not try to find best answers for individual statements. It tries to emulate reasoner behavior by matching the shape of the reasoning patterns. That is not to say we don't care about right answers, or that they don't matter. Those values are reported for our system. And if reasoning structure is matched well enough then correct answers should follow. But we believe that the reasoning distance results we report are far more compelling and tell a better story about what is happening, as well as indicate the future potential of this method.

Approach

We present a method for learning the structure of $\mathcal{EL}^+$ reasoning patterns using an LSTM that tests the questions posed above. In our system we avoid, as much as possible, embeddings or any distance adjustments that might help the system learn answers based on embedding similarity without first learning the reasoning structure. In fact, a primary feature of our network is its lack of complexity. We take a very simple logic, reason over knowledge bases in that logic, and then extract supports from the reasoning steps, mapping the reasoner supports back to sets of the original knowledge base axioms. This allows us to encode the input data in terms of only knowledge base statements. It also provides an intermediate answer that might improve results when provided to the system. This logic data is fed into three different LSTM architectures with identical input and output dimensionalities. One architecture, which we call "Deep", does not train with support data but has a hidden layer the same size as the supports we have defined. Another architecture, called "Piecewise", trains two separate half-systems, one with knowledge bases and one with supports from the reasoner. The last system, called "Flat", simply learns to map inputs directly to outputs for each reasoning step. We will discuss in detail each part of the system in the following sections, then provide some results.

$\mathcal{EL}^+$

$\mathcal{EL}^+$ is a lightweight and highly tractable description logic. A typical reasoning task in $\mathcal{EL}^+$ is a sequential process with a fixed endpoint, making it a perfect candidate for sequence learning. Unlike RDF, which reasons over instance data in triple format, $\mathcal{EL}^+$ reasoning occurs on the predicate level. Thus we will have to train the system to actually learn the reasoning patterns and logical structure of $\mathcal{EL}^+$ directly from encoded knowledge bases.

Syntax The signature Σ for $\mathcal{EL}^+$ is defined as:

$$\Sigma = \langle N_I, N_C, N_R \rangle \text{ with } N_I, N_C, N_R \text{ pairwise disjoint.}$$

N_I is a set of individual names.
N_C is a set of Concept names that includes $\top$.
N_R is a set of Role names.

Expressions in $\mathcal{EL}^+$ use the following grammar:

$$\begin{array}{lll} \mathbf{R} & ::= & N_R \\ \mathbf{C} & ::= & N_C \quad | \quad \mathbf{C} \sqcap \mathbf{C} \quad | \quad \exists \mathbf{R}.\mathbf{C} \end{array}$$

Semantics An interpretation I where $I = (\Delta^I, \cdot^I)$ maps N_I, N_C, N_R to elements, sets, and relations in Δ^I with function $\cdot^I$. For an interpretation I and $C(i), R(i) \in \Sigma$, the function $\cdot^I$ is defined in Table 1.

Table 1: $\mathcal{EL}^+$ Semantics

Description	Expression	Semantics
Individual	a	$a \in \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Concept	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential Restriction	$\exists R.C$	$\{ \ a \mid$ there is $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}} \ \}$
Concept Subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Role Chain	$R_1 \circ \cdots \circ R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \cdots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$

with $\circ$ signifying standard binary composition

Table 2: $\mathcal{EL}^+$ Completion Rules

(1)	$A \sqsubseteq C$	$C \sqsubseteq D$		$\models A \sqsubseteq D$
(2)	$A \sqsubseteq C_1$	$A \sqsubseteq C_2$	$C_1 \sqcap C_2 \sqsubseteq D$	$\models A \sqsubseteq D$
(3)	$A \sqsubseteq C$	$C \sqsubseteq \exists R.D$		$\models A \sqsubseteq \exists R.D$
(4)	$A \sqsubseteq \exists R.B$	$B \sqsubseteq C$	$\exists R.C \sqsubseteq D$	$\models A \sqsubseteq D$
(5)	$A \sqsubseteq \exists S.D$	$S \sqsubseteq R$		$\models A \sqsubseteq \exists R.D$
(6)	$A \sqsubseteq \exists R_1.C$	$C \sqsubseteq \exists R_2.D$	$R_1 \circ R_2 \sqsubseteq R$	$\models A \sqsubseteq \exists R.D$

Reasoning It is a well established result that any $\mathcal{EL}^+$ knowledge base has a least fixed point that can be determined by repeatedly applying a finite set of axioms that produce all entailments of a desired type.(Besold et al. 2017; Kazakov, Krötzsch, and Simančík 2012) In other words, we can say that reasoning in $\mathcal{EL}^+$ often amounts to a sequence of applications of a set of pattern-matching rules. One such set of rules, the set we have used in our experiment, is given in Table 2. The reasoning reaches *completion* when there are no new conclusions to be made. Because people are usually interested most in concept inclusions and restrictions, those are the types of statements we choose to include in our reasoning.

After the reasoning has finished we are able to recursively define supports for each conclusion the reasoner reaches. The first step, of course, only has supports from the knowledge base. After this step supports are determined by effectively running the reasoner in reverse, and replacing each statement that is not in the original knowledge base with a superset that is, as you can see by the colored substitutions in Table 3. When the reasoner proved the last statement it did not consider all of the supports, since it had already proved them. It used the new facts it had learned in the last iteration but we have drawn their supports back out so that we can define a fixed set of inputs from the knowledge base.

Synthetic Data

To provide sufficient training input to our system we provide a synthetic generation procedure that combines a structured forced-lower-bound reasoning sequence with a connected randomized knowledge base. This allows us to rapidly generate many normal semi-random $\mathcal{EL}^+$ knowledge bases of arbitrary reasoning difficulty. For this experiment we choose a moderate difficulty setting so that it can compare with non-synthetic data. An example of one iteration of the two-part sequence is provided in Figure 1. To ensure that the randomized statements do not shortcut this pattern, the random statements are generated in a nearly disjoint space and connected only to the initial seed term. This ensures that at least one element of the random space will also produce random entailments for the duration of the sequence, possibly longer. Our procedure also guarantees that each completion rule will be used at least once every iteration of the sequence so that all reasoning patterns can potentially be learned by the system.

seed_0 = C1 $\sqsubseteq$ C2	C2 $\sqsubseteq$ ∃R1.C3
C2 $\sqsubseteq$ C3	C1 $\sqsubseteq$ ∃R3.C4
C3 $\sqcap$ C4 $\sqsubseteq$ C5	C2 $\sqsubseteq$ ∃R1.C3
C1 $\sqsubseteq$ ∃R1.C1	∃R1.C2 $\sqsubseteq$ C4
C1 $\sqsubseteq$ ∃R2.C3	R1 $\sqsubseteq$ R2
C2 $\sqsubseteq$ ∃R2.C3	R1 $\circ$ R3 $\sqsubseteq$ R4

Which entails

Step 1:	Step 2:
C1 $\sqsubseteq$ C3	seed_1 = C1 $\sqsubseteq$ C5
C1 $\sqsubseteq$ C4	
C1 $\sqsubseteq$ ∃R1.C3	
C1 $\sqsubseteq$ ∃R2.C1	
C1 $\sqsubseteq$ ∃R4.C4	

Figure 1: First Iteration of Sequence

SNOMED

We have also imported data from the SNOMED 2012 ontology to ensure that our method is applicable to non-synthetic data. SNOMED is a widely-used, publicly available, ontol-

Table 3: Support Generation

	New Fact	Rule	Support
Step 1	$C1 \sqsubseteq C3$	(1)	$C1 \sqsubseteq C2, C2 \sqsubseteq C3$
	$C1 \sqsubseteq C4$	(4)	$C1 \sqsubseteq C2, C1 \sqsubseteq \exists R1.C1, \exists R1.C2 \sqsubseteq C4$
	$C1 \sqsubseteq \exists R1.C3$	(3)	$C1 \sqsubseteq C2, C2 \sqsubseteq \exists R1.C3$
	$C1 \sqsubseteq \exists R2.C1$	(5)	$C1 \sqsubseteq \exists R1.C1, R1 \sqsubseteq R2$
	$C1 \sqsubseteq \exists R4.C4$	(6)	$C1 \sqsubseteq \exists R1.C1, R1 \circ R3 \sqsubseteq R4, C1 \sqsubseteq \exists R3.C4$
Step 2	$C1 \sqsubseteq C5$	(2)	$C3 \sqcap C4 \sqsubseteq C5, C1 \sqsubseteq C2, C2 \sqsubseteq C3, C1 \sqsubseteq C2, C1 \sqsubseteq \exists R1.C1, \exists R1.C2 \sqsubseteq C4$

ogy of medical terms and relationships.(De Silva et al. 2011) SNOMED 2012 has 39392 logical axioms, some of which are complex, but this can be normalized in constant time to a logically equivalent set of 124,428 axioms. It is expressed in $\mathcal{EL}^{++}$, and newer versions utilize this complexity more liberally. But the 2012 version contains exactly one statement that is not in $\mathcal{EL}^{+}$ after normalization. We feel confident that omitting $\top \sqsubseteq \exists$topPartOf.Self does not constitute a substantial loss in the ontology.

Because SNOMED is so large, we sample it to obtain connected subsets that also produce a minimum number of reasoning steps. We require that the samples be connected because any normal knowledge base is connected, and it improves the chances that the statements will have entailments. Often, however, random connected samples do not entail anything, and this is also unusual in a normal ontology, so we require a minimum amount of reasoning activity in the samples as well. The reasoning task for SNOMED is more unbalanced than for the synthetic data. It is equally trivial for the reasoner to solve either knowledge base type. However, we observe that random connected sampling tends to favor application of rules 3, 5, and 6 much more heavily than others so the system will have a more difficult time learning the overall reasoning patterns. This imbalance is likely an artifact from SNOMED because it seems to recur in different sample sizes with different settings, though we acknowledge that it could be correlated somehow with the sample generation procedure.

LSTM

LSTMs are type of recurrent neural network that work well with data that has long-term dependencies.(Hochreiter and Schmidhuber 1997) We choose to use an LSTM to learn the reasoning patterns of $\mathcal{EL}^{+}$ because LSTMs are designed to learn sequence patterns. The network is kept as simple as possible to achieve the outputs we require and maximize transparency. For the piecewise system, depicted in Figure 2, we use two separate flat single LSTMs that have the number of neurons matching first the support shape, and later the output shape. The deep system shown in Figure 3 is constructed to match this shape so that the two will be comparable, it simply stacks the LSTMs together so that it can train without supports. We also train a flat system, shown in Figure 4, that has the same input and output shape but lacks the internal supports. We have done this three ways because we can compare the behavior of the systems with support

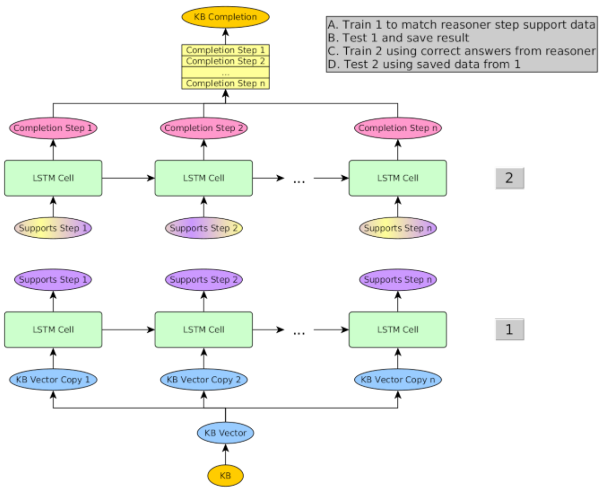


Figure 2: Piecewise Architecture

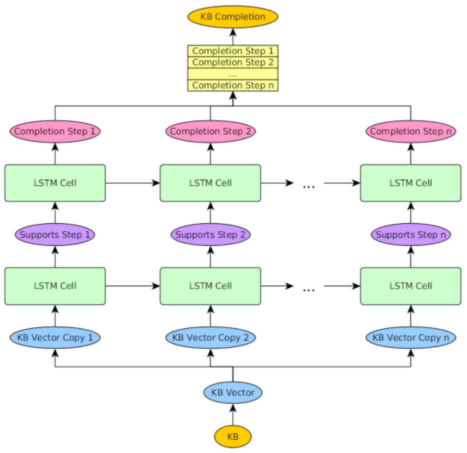


Figure 3: Deep Architecture

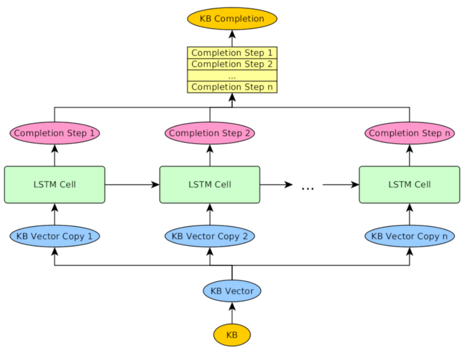


Figure 4: Flat Architecture

against the flat system as a baseline, and then see whether the piecewise training is helping. All designs treat the reasoning steps as the sequence to learn, so the number of LSTM cells is defined by the maximum reasoning sequence length. We train our LSTMs using regression on mean squared error to minimize the distances between predicted answers and correct answers.

Overall System

The individual parts of our system are not themselves novel. Our result, however, lies rather in the unexpected success of this unconventional general approach we have applied. One of the crucial factors in making this work is the way in which we have translated the data from its logical format into something a neural network can understand. In the following sections we will discuss how our system transforms data from a knowledge base format to an LSTM format and back again.

Statement Encoding Every data set we use has a fixed number of names that it can contain for both roles and concepts, and every concept or role has an arbitrary number name identifier. For the synthetic data these numbers are randomly generated. For SNOMED data, all of the labels are stripped and the concept and role numbers are shuffled around and substituted with random integers to remove any possibility of the names injecting a learning bias. These labels are remembered for later retrieval for output but the reasoner and LSTM do not see them. It can seem occasionally like there is a bias in the naming if output for SNOMED data is inspected because all the labels seem related. This is merely a result of our sampling technique which requires connected statements, so it is not a concern.

Knowledge bases have a maximum number of role and concept names that are used to scale all of the integer values for names into a range of $[-1, 1]$. To enforce disjointness of concepts and roles, we map all concepts to $(0, 1]$, all roles to $[-1, 0)$. Each of the six possible normalized axiom forms is encoded into a 4-tuple based on the logical encodings defined in Table 4.

We would like to emphasize the distinction between our method, which we prefer to call an encoding, and a traditional embedding. This encoding adds no additional information to the names beyond the transformation from integer

Table 4: Translation Rules

KB statement		Vectorization
CX $\sqsubseteq$ CY	$\rightarrow$	$[\,0.0,\ \frac{X}{c},\ \frac{Y}{c},\ 0.0\,]$
CX $\sqcap$ CY $\sqsubseteq$ CZ	$\rightarrow$	$[\,\frac{X}{c},\ \frac{Y}{c},\ \frac{Z}{c},\ 0.0\,]$
CX $\sqsubseteq$ $\exists$RY.CZ	$\rightarrow$	$[0.0,\ \frac{X}{c},\ \frac{-Y}{r},\ \frac{Z}{c}]$
$\exists$RX.CY $\sqsubseteq$ CZ	$\rightarrow$	$[\frac{-X}{r},\ \frac{Y}{c},\ \frac{Z}{c},\ 0.0]$
RX $\sqsubseteq$ RY	$\rightarrow$	$[0.0,\ \frac{-X}{r},\ \frac{-Y}{r}, 0.0]$
RX $\circ$ RY $\sqsubseteq$ RZ	$\rightarrow$	$[\frac{-X}{r},\ \frac{-Y}{r},\ \frac{-Z}{r}, 0.0]$

c = Number of Possible Concept Names
r = Number of Possible Role Names

to scaled floating point number. It can be reversed, with a slight loss of precision due to integer conversion, quite directly. The semantics of each encoded predicate name with regards to its respective knowledge base is structural in relation to its position in the 4-tuple, just like it would be in an $\mathcal{EL}^+$ axiom. And its semantics is not at all related to the other non-equal predicate names within the same 4-tuple.

Knowledge Base Encoding In order to input knowledge bases into the LSTM we first apply the encoding defined in the previous section to each of its statements. Then we concatenate each of these encodings end-to-end to obtain a much longer vector. For instance,

$$\text{C1} \sqsubseteq \exists\text{R1.C2, R1} \sqsubseteq \text{R2}$$

might translate to

$$[0.0, 0.5, -0.5, 1.0, 0.0, -0.5, -1.0, 0.0].$$

This knowledge base vector is then copied the same number of times as there are reasoning steps and stuck together once again along a new dimension. For an experiment this is done hundreds or thousands of times, and these fill up the tensor that is given to the LSTM to learn. Because some of the randomness can cause ragged data, any tensor dimensions that are not the same length as the maximum are padded with zeros.

Results

Viewed in terms of the questions posed at the beginning of this paper we feel that our results are solid, and they validate a reexamination of how we should approach integrating logic and neural networks. We have attempted to create a system that emulates a reasoning task rather than reasoning output.

If we examine the example output from the synthetic data inputs in Table 5, it is clear that it is getting very close to many correct answers. When it misses, it still appears to be learning the shape, and this makes us optimistic about its future potential. The SNOMED predictions are much more dense and do not fit well into a table, but we have included a few good examples with the original data labels

Table 5: Example Synthetic Output

	Correct Answer	Predicted Answer
Step 0	C9 ⊑ C11	C8 ⊑ C9
	C2 ⊑ C10	C1 ⊑ C9
	C9 ⊑ C12	C8 ⊑ C9
	C7 ⊑ C6	C8 ⊑ ∃R4.C9
	C9 ⊑ ∃R4.C11	C1 ⊑ ∃R5.C9
	C2 ⊑ ∃R4.C9	C8 ⊑ ∃R4.C9
	C9 ⊑ ∃R5.C9	C9 ⊑ ∃R5.C9
	C2 ⊑ ∃R5.C11	
	C9 ⊑ ∃R7.C12	
	C2 ⊑ ∃R6.C12	
Step 1	C9 ⊑ C13	C8 ⊑ C12
	C2 ⊑ C11	C2 ⊑ C10
	C2 ⊑ C12	C1 ⊑ C11
	C2 ⊑ ∃R4.C11	C1 ⊑ ∃R3.C12
	C2 ⊑ ∃R5.C9	C1 ⊑ ∃R4.C8
	C2 ⊑ ∃R7.C12	
Step 2	C2 ⊑ C13	C1 ⊑ C12

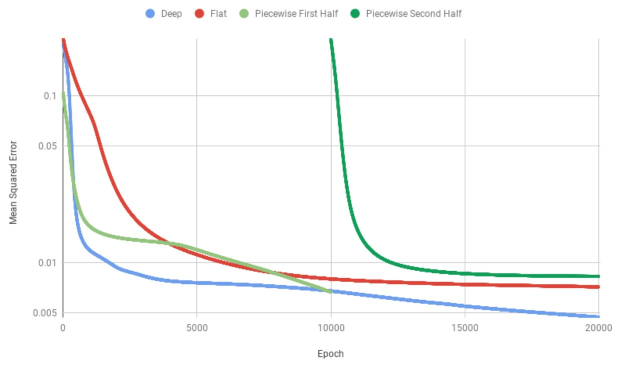


Figure 5: Synthetic Training

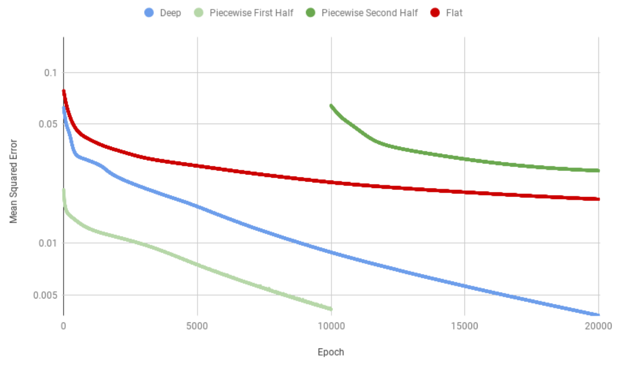


Figure 6: SNOMED Training

Table 6: Example SNOMED Outputs

Correct Answer	C6 ⊑ ∃R4.C1
Meaning	if something is a zone of superficial fascia, then there is a subcutaneous tissue that it is PartOf
Prediction	C6 ⊑ ∃R4.C3
Meaning	if something is a zone of superficial fascia, then there is a subcutaneous tissue of palmar area of little finger that it is PartOf
Correct Answer	C8 ⊑ ∃R3.C2
Meaning	if something is a infrapubic region of pelvis, then there is a perineum that it is PartOf
Prediction	C9 ⊑ ∃R3.C2
Meaning	if something is a zone of integument, then there is a perineum that it is PartOf

translated into English sentences. Because there are so many near-misses we include edit distance evaluations that better capture the degree to which each predicted statement misses what it should have been.

It is interesting to note that by comparing Table 7 with Table 8 we can see that on the much harder SNOMED data the deep system *appears* to have a better result because of the higher F1 score, but the average edit distance, which is our preferred alternative measure for evaluation, is not obviously correlated with the F1-score. This confirms that our fifth research question was appropriate and that F1-score might not be sufficient to evaluate whether or not we can learn the shape of reasoning patterns.

A cause for the discrepancy may be the higher training difficulty for reaching the completion versus reaching the supports in the SNOMED data, which you can see in Figures 5 and 6. We can speculate on the degree to which various factors are contributing to this, but importantly, the fact that this discussion is even possible in the first place is a confirmation that we can answer the fourth research question in the affirmative.

Another interesting result can be seen by examining the charts in Figures 7 - 12. As we have noticed before, the Deep architecture performs quite well on the synthetic data and is quite resistant to increased levels of noise. The flat sys-

Table 7: Average Precision Recall and F1-score For each Distance Evaluation

	Atomic Levenshtein Distance			Character Levenshtein Distance			Predicate Distance		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
	Synthetic Data								
Piecewise Prediction	0.138663	0.142208	0.140412	0.138663	0.142208	0.140412	0.138646	0.141923	0.140264
Deep Prediction	**0.154398**	**0.156056**	**0.155222**	**0.154398**	**0.156056**	**0.155222**	**0.154258**	**0.155736**	**0.154993**
Flat Prediction	0.140410	0.142976	0.141681	0.140410	0.142976	0.141681	0.140375	0.142687	0.141521
Random Prediction	0.010951	0.0200518	0.014166	0.006833	0.012401	0.008811	0.004352	0.007908	0.007908
	SNOMED Data								
Piecewise Prediction	0.010530	0.013554	0.011845	0.010530	0.013554	0.011845	0.010521	0.013554	0.011839
Deep Prediction	**0.015983**	0.0172811	**0.016595**	**0.015983**	0.017281	**0.016595**	**0.015614**	0.017281	**0.016396**
Flat Prediction	0.014414	**0.018300**	0.016112	0.0144140	**0.018300**	0.016112	0.013495	**0.018300**	0.015525
Random Prediction	0.002807	0.006803	0.003975	0.001433	0.003444	0.002023	0.001769	0.004281	0.002504

Table 8: Average Statement Edit Distances with Reasoner

	Atomic Levenshtein Distance			Character Levenshtein Distance			Predicate Distance		
	From	To	Average	From	To	Average	From	To	Average
	Synthetic Data								
Piecewise Prediction	1.336599	1.687640	1.512119	1.533115	1.812006	1.672560	2.633427	4.587382	3.610404
Deep Prediction	**1.256940**	**1.507150**	**1.382045**	**1.454787**	**1.559751**	**1.507269**	**2.504496**	**3.552074**	**3.028285**
Flat Prediction	1.344946	1.584674	1.464810	1.586281	1.660409	1.623345	2.517655	3.739770	3.128713
Random Prediction	1.598016	1.906369	1.752192	1.970604	1.289533	1.630068	5.467918	10.57324	8.020583
	SNOMED Data								
Piecewise Prediction	1.704931	**2.686562**	**2.195746**	2.016249	2.862737	2.439493	6.556592	**5.857769**	6.207181
Deep Prediction	1.759633	3.052080	2.405857	2.027190	3.328850	2.678020	**4.577427**	6.179389	**5.378408**
Flat Prediction	**1.691738**	2.769542	2.230640	**1.948757**	2.991328	2.470042	5.548226	6.665659	6.106942
Random Prediction	1.814656	3.599629	2.707143	2.094682	**1.621700**	**1.858191**	5.169093	12.392325	8.780709

tem gets similar results, but slightly worse. Surprisingly, the piecewise model starts out mostly the same as the deep system on synthetic data but then very quickly falters. When we examine the results for the harder SNOMED data, however, the opposite seems to be happening. The deep and flat systems still track each other. But for the unbalanced data the deep system crosses into the random range by 30-50% corruption, while the piecewise system is able to skirt just above random for the Character and Predicate distance functions until around 80-90%.

Methodology

Our system is trained using randomized 10-fold cross validation to 20000 epochs on the deep and flat systems and 10000 epochs each for the parts of the piecewise system at a learning rate of 0.0001. The results from every fold are saved and we report the average of all of the runs. Often there are better and worse runs in any given 10-fold validation but the averages across multiple cross-validations with the same settings remain basically the same. 10-fold cross-validations are performed with an extra comparison against data that has been corrupted at a fixed rate, starting with zero percent probability of corruption and moving upwards by ten percent each time. The data in Tables 7 and 8 is from the comparison with no corruption, so the error data is not reported (it is identical with reasoner answers).

To perform our evaluations we use three unique distance measurements. We have a naive "Character" Levenshtein distance function that takes two unaltered knowledge base statement strings and computes their edit distance.(Wikibooks contributors 2019 accessed November 19 2019) However, because some names in the namespace are one digit numbers and other names are two digit numbers, we include a modified version of this function, called "Atomic", that uniformly substitutes all two digit numbers in the strings with symbols that do not occur in either. Since there cannot be more than eight unique numbers in two decoded strings there are no issues with finding enough new symbols. By doing the substitutions we can see the impact that the number digits were having on the edits from the Atomic Levenshtein distance. Finally we devise a distance function that is based on our encoding scheme. The Predicate Distance method disassembles each string into only its predicates. Then, for each position in the 4-tuple, a distance is calculated that yields zero for perfect matches, absolute value of guessed number - actual number for correct Class and Role guesses, and guessed number + actual number for incorrect Class and Role matches. So, for instance, guessing C1 when the answer is C2 will yield a Predicate Distance of 1, while a guess of R2 for a correct answer of C15 will yield 17. Though this method is specific to our unique encoding, we believe it detects good and bad results quite well because perfect hits are 0, close misses are penalized a little, and large misses are penalized a lot.

For each method we take every statement in a knowledge base completion and compare it with the best match in the reasoner answer, random answers, and corrupted knowledge base answers. While we compute these distances we are able to obtain precision, recall, and F1-score by counting the the number of times the distance returns zero and treating the statement predictions as classifications. Each time the system runs it can make any number of predictions, from zero to the maximum size of the output tensor. This means that, although the predictions and reasoner are usually around the same size and those comparisons are fine, we have to generate random data to compare against that is as big as could conceivably be needed by the system. Any artificial shaping decisions we made to compensate for the variations between runs would invariably introduce their own bias in how we selected them. Thus the need to use the biggest possible random data to compare against means the precision, recall, and F1-score for random are low.

Conclusion

In this experiment demonstrate that $\mathcal{EL}^+$ reasoning can be emulated with an LSTM. Although our example is merely a prototype, we can definitively say yes to the first research question, a neural network can be trained to emulate completion reasoning behavior. And because we have answered this first research question without using name adjustments or embeddings, so we can also answer yes to the second and third questions. We have already discussed questions four and five.

Future Work

There are a number of potential improvements that could build on the work presented here. A few of the choices we made might have had an impact of the results of the study and we are curious if alternate strategies could improve the way that the system learns reasoning patterns. We might have chosen a 3-tuple encoding pattern that was more dense but did not mirror the shape of the statements symmetrically. We also considered adding the statements as an additional dimension rather than concatenating them all together. This may have made it easier for the system to distinguish individual statements but would also add a lot of complexity in the extra dimension. We experimented with different neuron types like gated recurrent unit (GRU) and basic RNN, though these had no significant impact on the results. The type of neuron seems to have less effect than the overall structure of the network. It would be interesting to see if different ways of segmenting a network, besides just along the supports as we have defined them, could improve results.

Along with these ideas to improve our current system, we also speculate that with some effort we should be able to accomplish basically the same type of thing with a more expressive logic and a new encoding. $\mathcal{EL}^{++}$ for instance would be an easy first attempt since it is so similar to $\mathcal{EL}^+$. Though we should be able to adapt this strategy for any logic that uses completion rules for reasoning. Whatever we choose to do with it, we are optimistic that we can use this strategy to advance understanding of how logic and neural networks can work together.

Testing Environment

All testing was done on a computer running Ubuntu 19.10 64-bit with an Intel Core i7-9700K CPU@3.60GHz x 8, 47.1 GiB DDR4, and a GeForce GTX 1060 6GB/PCIe/SSE2. Source code and experiment data is available on GitHub https://github.com/aaronEberhart/PySynGenReas.

Acknowledgements This work was partially supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386, and by the National Science Foundation under award number 1936677.

References

Besold, T. R.; d'Avila Garcez, A. S.; Bader, S.; Bowman, H.; Domingos, P. M.; Hitzler, P.; Kühnberger, K.; Lamb, L. C.; Lowd, D.; Lima, P. M. V.; de Penning, L.; Pinkas, G.; Poon, H.; and Zaverucha, G. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR* abs/1711.03902.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, 2787–2795.

Das, R.; Neelakantan, A.; Belanger, D.; and McCallum, A. 2016. Chains of reasoning over entities, relations, and text using recurrent neural networks. *CoRR* abs/1607.01426.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; and McCallum, A. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*.

De Silva, T. S.; MacDonald, D.; Paterson, G.; Sikdar, K. C.; and Cochrane, B. 2011. Systematized nomenclature of medicine clinical terms (snomed ct) to represent computed tomography procedures. *Comput. Methods Prog. Biomed.* 101(3):324–329.

Ebrahimi, M.; Sarker, M. K.; Bianchi, F.; Xie, N.; Doran, D.; and Hitzler, P. 2018. Reasoning over rdf knowledge bases using deep learning. *arXiv preprint arXiv:1811.04132*.

Hochreiter, S., and Schmidhuber, J. 1997. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, 473–479.

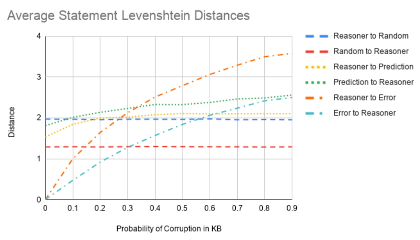
Kazakov, Y.; Krötzsch, M.; and Simančík, F. 2012. Elk: a reasoner for owl el ontologies. *System Description*.

Kulmanov, M.; Liu-Wei, W.; Yan, Y.; and Hoehndorf, R. 2019. El embeddings: Geometric construction of models for the description logic el++. *arXiv preprint arXiv:1902.10499*.

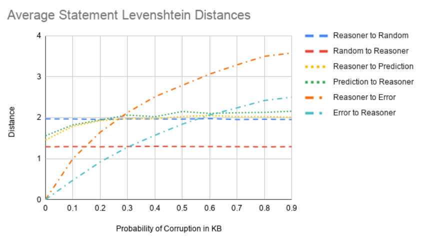
Makni, B., and Hendler, J. 2018. *Deep Learning for Noise-tolerant RDFS Reasoning*. Ph.D. Dissertation, Rensselaer Polytechnic Institute.

Wikibooks contributors. 2019 (accessed November 19, 2019). Algorithm implementation/strings/levenshtein distance.

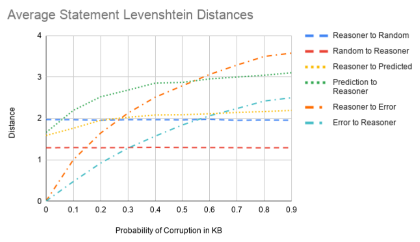
Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*.



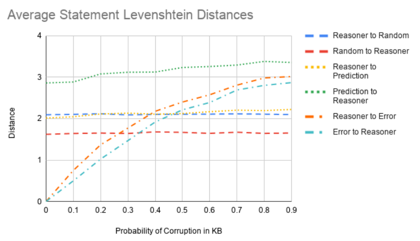
(a) Synthetic Data Piecewise Architecture



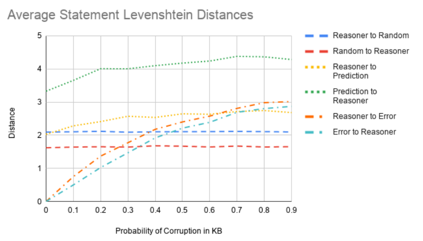
(b) Synthetic Data Deep Architecture



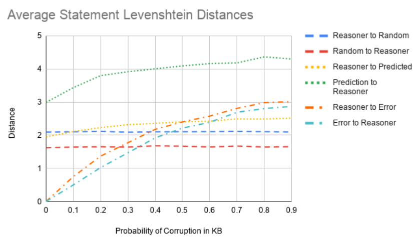
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

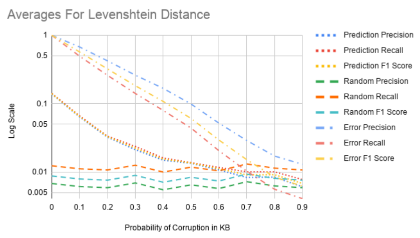


(e) SNOMED Data Deep Architecture

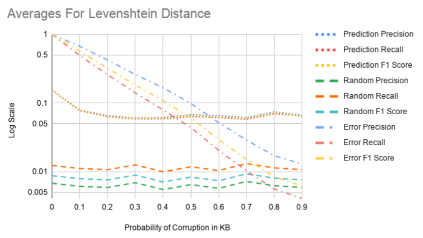


(f) SNOMED Data Flat Architecture

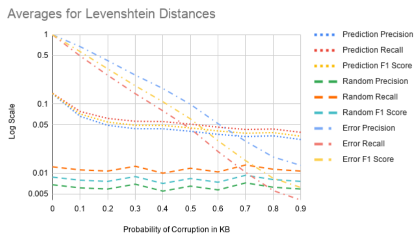
Figure 7: Character Levenshtein Distance



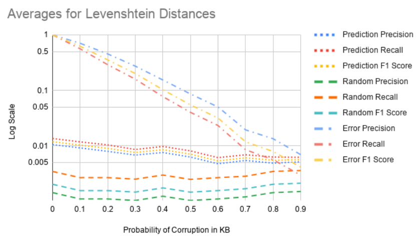
(a) Synthetic Data Piecewise Architecture



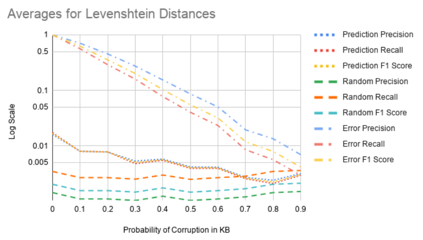
(b) Synthetic Data Deep Architecture



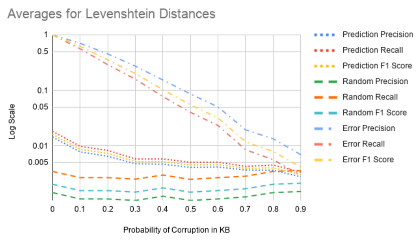
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

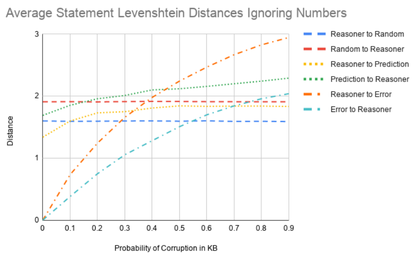


(e) SNOMED Data Deep Architecture

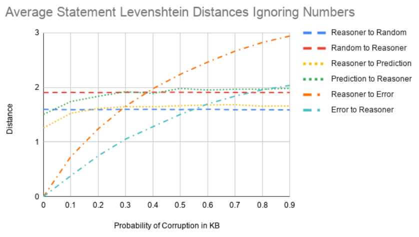


(f) SNOMED Data Flat Architecture

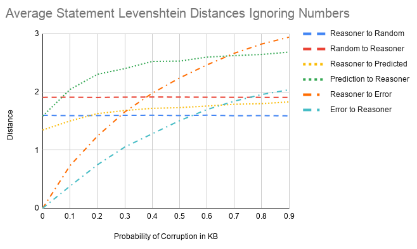
Figure 8: Character Levenshtein Distance Precision, Recall, and F1-score



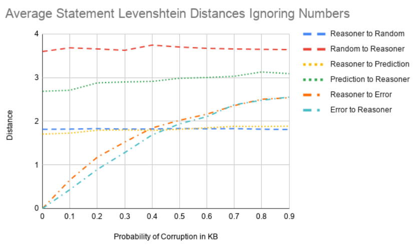
(a) Synthetic Data Piecewise Architecture



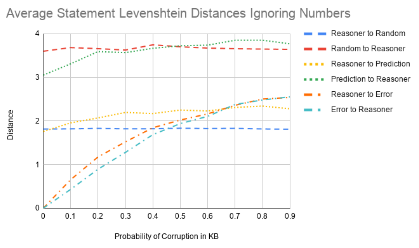
(b) Synthetic Data Deep Architecture



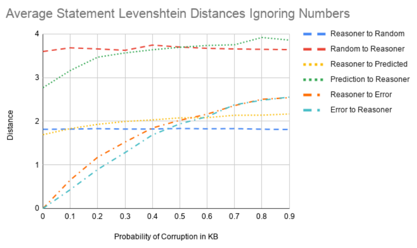
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

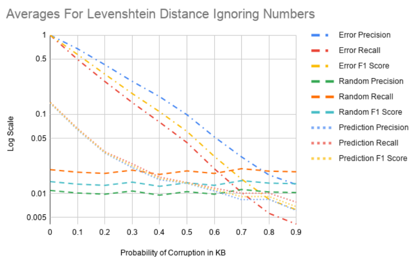


(e) SNOMED Data Deep Architecture

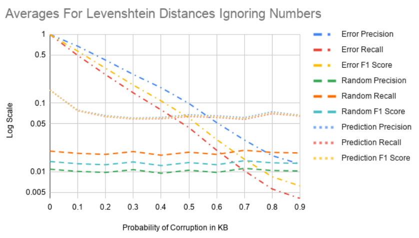


(f) SNOMED Data Flat Architecture

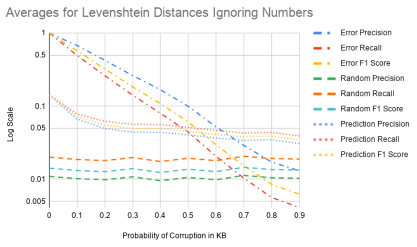
Figure 9: Atomic Levenshtein Distance



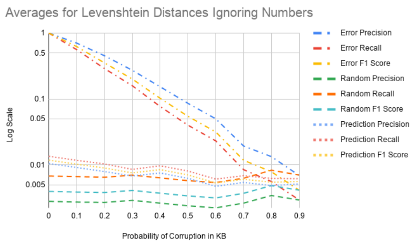
(a) Synthetic Data Piecewise Architecture



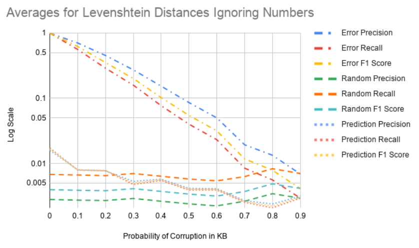
(b) Synthetic Data Deep Architecture



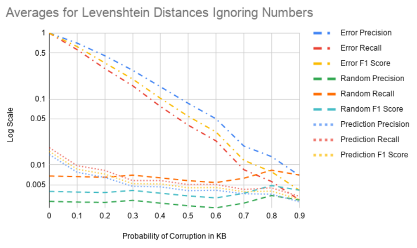
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

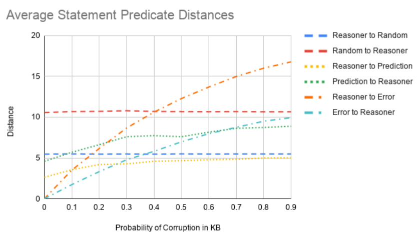


(e) SNOMED Data Deep Architecture

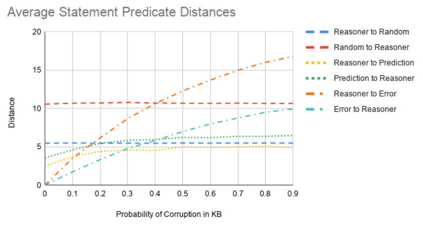


(f) SNOMED Data Flat Architecture

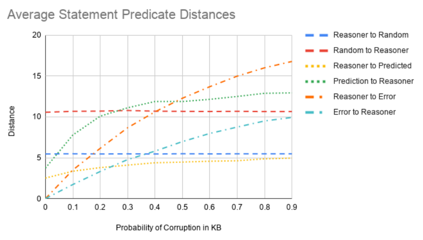
Figure 10: Atomic Levenshtein Distance Precision, Recall, and F1-score



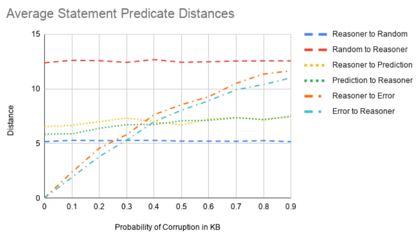
(a) Synthetic Data Piecewise Architecture



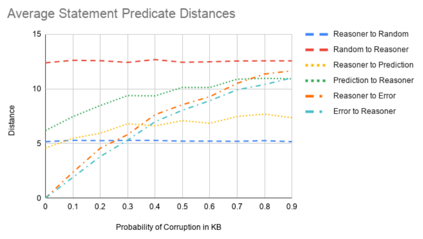
(b) Synthetic Data Deep Architecture



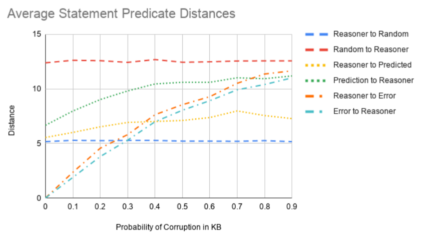
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

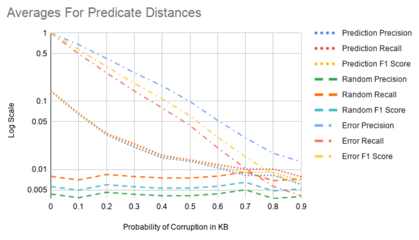


(e) SNOMED Data Deep Architecture

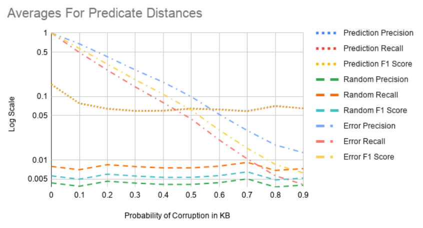


(f) SNOMED Data Flat Architecture

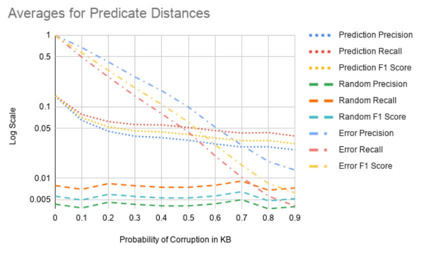
Figure 11: Predicate Distance



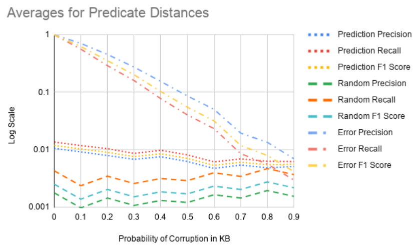
(a) Synthetic Data Piecewise Architecture



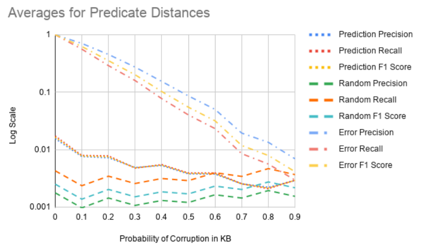
(b) Synthetic Data Deep Architecture



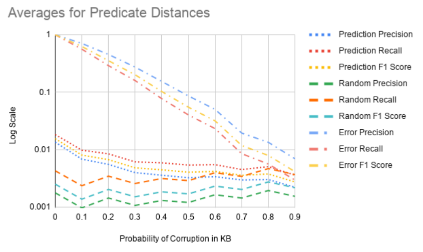
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture



(e) SNOMED Data Deep Architecture



(f) SNOMED Data Flat Architecture

Figure 12: Predicate Distance Precision, Recall, and F1-score