CORGI: Common-sense Reasoning by Instruction

Paper Id: 9564

Abstract

Currently, almost all reasoning engines rely on querying a large knowledge base containing background knowledge of the world. However, it is known that even the largest knowledge bases gathered to date are incomplete. This is expected, since there is currently no technology that has the capacity to store and query the sum total of all human knowledge. Therefore, in this paper, we propose CORGI: a **CO**mmon-sense **R**easonin**G** by **I**nstruction strategy for completing reasoning tasks where only a small amount of background knowledge is available . When faced with missing knowledge, CORGI initiates a conversation with a human user and extracts that knowledge from them just-in-time. Our contributions are twofold. First, we propose a commonsense reasoning method through conversation. Second, in order to deal with variations of natural language in conversation, we propose to combine symbolic AI with neural approaches and present a "soft" reasoning engine that uses the best of both worlds.

Introduction

Justice, Pence, and Beckman state in their book that "Children develop their knowledge of the world around them as they interact with their environment directly and indirectly. The direct experiences children have in their homes, schools and communities certainly provide the greatest amount of input to the world knowledge base." (Justice, Pence, and Beckman 2005). This knowledge arises from both physical and conversational interactions. In this paper, we test the hypothesis that just like a human child, machines need interaction to acquire world knowledge and develop commonsense reasoning abilities, and we study the effect of conversational interactions on this knowledge acquisition. Most of the literature on commonsense reasoning relies on extracting the largest possible snapshot of world knowledge and either query it or propose automated knowledge base completion methods for it. We argue that it is necessary to equip reasoning engines with an interaction strategy facilitating the extraction of just-in-time information needed for reasoning . In this paper, we take up this grand goal, and although we do not solve the whole challenge, we take the first steps needed for addressing it.

We believe that this is the right time for this proposal specifically since conversational agents such as Siri, Google home, Alexa and Cortana among others are starting to enter our daily lives. Therefore, it is plausible to assume that such agents have access to conversation with a human for extracting commonsense knowledge. In this paper, we work with the Learning by Instruction Agent (LIA) (Azaria, Krishnamurthy, and Mitchell 2016; Labutov, Srivastava, and Mitchell 2018) and develop a commonsense reasoning system for her called CORGI (**CO**mmonsense **R**easonin**G** by **I**nstruction). In what follows, we present our definition of commonsense reasoning for LIA after briefly introducing her.

LIA is an intelligent agent that operates on a user's smartphone. End users add new functionalities to LIA through verbal instructions and teach her how to perform new tasks. For example the user can tell LIA, "whenever it snows at night, wake me up 30 minutes early". If LIA does not understand how to perform this task, she will ask the user to instruct her by breaking the task down into a set of steps in a teaching session. In this case, the user can say, "(first) open the weather app, (second) see if the night weather condition is snow, (third) if true then adjust my alarm to 30 minutes earlier". After this teaching session, LIA can perform this task.

One phenomena we have noticed in collecting these types of "Whenever S occurs, then do A" instructions is that people often *underspecify* the precondition S. For example, one instructor might want to wake up early when it snows because they are concerned about getting to work on time. For this user, the implied precondition is not really "whenever it snows," but instead "whenever it snows enough to cause traffic slowdowns, and it's a workday." The point is that people often fail to specify the detailed conditions, perhaps because they are used to speaking to other people who possess the common sense needed to infer the more specific intent of the speaker.

Our goal for LIA is to use background commonsense knowledge to reason about the user's more specific intent, and to discuss this with the user in order to create the correct preconditions for the recommended action. Therefore, we assume LIA can obtain statements from the user that fit the logical template "Whenever S occurs, do A because I want

to achieve goal G."[1] For example consider the following two statements:

- Whenever it snows at night, wake me up 30 minutes early because I don't want to be late to work
- Whenever it snows at night, wake me up 30 minutes early because I have never seen the snow before

Note that in the first statement, the user will not want to wake up early on a weekend or a holiday (assuming that they do not work then) whereas in the second scenario, the user will want to wake up early regardless of the date in order to see snow for the first time – but might not want to wake up early after she has seen snow one time.

In CORGI, the role of commonsense reasoning is to derive the intended condition to use in place of the stated S given an "If S then do A because G" statement from the user. Its general approach is to derive an explanation of how action A, performed in state S will achieve goal G, and then to derive the intended precondition S by collecting the preconditions on S that allow this explanation to hold. CORGI has access to a limited amount of general background knowledge about the world, represented in a logic programming language. Reasoning reduces to using this background knowledge to perform multi-hop logical inference. If no reasoning path is found, CORGI initiates a conversation with the user to extract relevant background knowledge and adds it to its underlying understanding of the world. This newly acquired background knowledge will be used in future user interactions with CORGI. In essence, we are performing knowledge base completion through conversation, on a need-driven basis. Note that in earlier work Hixon, Clark, and Hajishirzi perform relation extraction using human interaction for question answering. Although the general idea of using human interaction is similar to our proposal, the information extraction method and the problem studied in Hixon, Clark, and Hajishirzi's work differs from our setting.

CORGI's main reasoning component is the multi-hop inference system. Since the knowledge is represented in a logical programming language, the underlying inference algorithm is backward chaining. However, backward chaining in its traditional form is not robust to variations in natural language. This is specifically of importance since CORGI allows open-domain dialog with the user to reduce the startup cost of the user having to learn a specific grammar or vocabulary. Therefore, there is no parsing algorithm to resolve these variations. For example, in traditional backward chaining, the statements "if the forecast is snow tonight" and "if the weather is snowy tonight" are thought of as two different statements whereas we want them both to map to the same representation. In order to address this, we propose a "soft backward chaining" algorithm that learns continuous representations or embeddings of the logical statements in the background knowledge. This will allow CORGI to indicate the equivalence of semantically similar statements based on the distance of their learned representations in the vector space. This soft backward chaining allows us to bridge a gap between symbolic AI and neural approaches using the best of both worlds.

[1]Note in LIA's conversational setting, if the user gives an instruction of the form "Whenever S occurs, do A." and omits the reason, then LIA can simply respond "Why do you want to do that?" in order to prompt for the missing reason G.

Related Work

Due to advances in natural language processing and the development of recent intelligent assistants, learning through user interaction such as verbal instruction or demonstration has gained momentum in the past few years (Azaria, Krishnamurthy, and Mitchell 2016; Labutov, Srivastava, and Mitchell 2018; Srivastava 2018; Srivastava, Labutov, and Mitchell 2017; Srivastava, Azaria, and Mitchell 2017; Li et al. 2018; 2017; Li, Azaria, and Myers 2017). The reinforcement learning community has also seen recent interest in using natural language statements as instruction (Hu et al. 2019; Luketina et al. 2019; Wang, Liang, and Manning 2016)

The literature on commonsense reasoning dates back to the very beginning of the field of artificial intelligence and is studied in several contexts. One of these is the literature on knowledge base construction and completion where the goal is to construct a large commonsense knowledge base and develop automated completion methods for it (Bosselut et al. 2019). Serafini and Garcez perform logical reasoning using neural networks for knowledge base completion. Another line of work is on "if-then" reasoning (Sap et al. 2018; Rashkin et al. 2018). In our work, in contrast to these studies, we start from a small amount of knowledge. We hypothesize that since no computing system has the capacity to store the sum total of all human commonsense knowledge, we need to develop methods that extract commonsense knowledge on a need-driven basis through conversation. Closest in nature to our proposal is Hixon, Clark, and Hajishirzi's work on relation extraction through conversation for question answering.

Commonsense reasoning is also studied in the question answering and visual question answering (VQA) literature. Saeidi et al. perform question answering for a dialog assistant that attempts to understand the context of the dialog and initiates clarifying question with the user to answer questions effectively. The literature on commonsense reasoning for visual question answering is also vast (Yi et al. 2018; Wu et al. 2018). Hudson and Manning study VQA using end-to-end differentiable methods.

CORGI's underlying reasoning strategy finds a chain of reasoning in a given commonsense knowledge base represented in Prolog. Chain of reasoning can be performed through logical inference and there are several interesting works that attempt to bridge the gap between symbolic AI and the recent advances in AI and machine learning (Liang et al. 2018; Chen et al. 2019; Zhou 2019; Wang et al. 2019; Evans and Grefenstette 2018). TensorLog (Cohen 2016) converts a first-order logical database into a factor graph and proposes a differentiable strategy for belief propagation over the graph. Contrary to our approach, TensorLog does not learn distributed representations for the logical rules. These representations are used by CORGI for combating natural language variations. The use of Prolog for performing multi-hop logical reasoning has been studied in (Rocktäschel and Riedel 2017; Weber et al. 2019) where neural networks are integrated into Prolog to perform soft unification in backward

chaining. In these works, the authors propose differntiable "AND" and "OR" operations that convert unification to a differentiable process. In contrast, we propose to use the proof trace of a program and learn the proving strategy from data. Our method is end-to-end differentiable and is an adaptation of the neural programmer interpreter (Reed and De Freitas 2015) that uses the trace of a program to learn program execution.

Problem Definition

In this paper, we would like to use commonsense background knowledge to reason about a user's specific intent and to engage in a conversation with the user to complete the background knowledge through this interaction. We assume the the user's utterance follows the general format "if $\langle$ state $S(X)\rangle$ then $\langle$ perform action $A(X)\rangle$ because $\langle$ I want to achieve goal $G(X)\rangle$". For example, "If it snows at night then wake me up early because I want to see the snow". We refer to the if statement as the state $S(X)$, the then condition as the action $A(X)$ and the because statement as the goal $G(X)$. The statement can then be read as: if state $S(X)$ is true, then perform action $A(X)$ because I want to achieve goal $G(X)$.

Moreover, we assume that we have access to a small amount of background knowledge K that we will use to reason about the given statements. Our background knowledge is represented in Prolog (Colmerauer 1990) in terms of logical rules.

Commonsense reasoning is then defined as the problem of "proving" the goal $G(X)$ against the system's background knowledge K such that the state $S(X)$ and action $A(X)$ are in the proof trace. Specifically

$$S(X) \cap A(X) \cap K \rightarrow G(X), \tag{1}$$

Given this problem setup we reason about the statement by following the proof path that leads to a valid proof for the goal. For example, the reasoning path for the example "If it snows at night then wake me up early because I get to work on time." is

1 if it snows more than 2 inches, then there will be traffic

2 if there is traffic then my commute time to work increases

3 if my commute time to work increases then i need to leave the house earlier to get to work on time

4 I will leave the house earlier if I wake up earlier.

Our goal is to find such a reasoning path for an input statement given the knowledge base and through engaging in a conversation with the user. In the above example, CORGI might already know items 1, 2 and 4 and not know how snow is relevant to getting on time to work. In this case, CORGI will ask the user how getting to work on time is relevant to the snow and hope to extract statement 3 from the user.

Prolog background and Notation

Before delving into the details of our proposed solution, let us introduce some notations and background. As stated earlier, we assume that CORGI has access to a small amount of background knowledge, K. In our study, K consists of logical rules represented in Prolog. In other words, our background knowledge is a Prolog program. Prolog is a declarative logical programming language in which programs consist of relations, which are a generalization of functions in functional programming. We have developed a modified version of Prolog which has been augmented to support several special features (types, soft-matched predicates and atoms, etc) required by our model.

In Prolog, all the data including a Prolog program are represented as Prolog terms. A Prolog program consists of a set of predicates. A predicate has a name (functor) and N arguments where $N \geq 0$. N is referred to as the arity of the predicate. for example, a predicate with functor name F and arity N is represented as $F(T_1, \ldots, T_N)$ where T_i for $i \in [1, N]$ are the arguments that are arbitrary Prolog terms. A prolog term is either an atom, a variable or a compound term. A predicate defines a relationship between its arguments. For example, isBefore(monday, tuesday) indicates that the relationship between Monday and Tuesday is that, one is before the other.

A predicate is defined by a set of clauses. A clause is either a Prolog *fact* or a Prolog *rule*. A Prolog rule is of the following format

$$\text{Head} :\!- \text{ Body}. \tag{2}$$

where the Head is a predicate and the body is a conjunction of predicates. The above rule is an if-then statement that is read as *if* the Body holds *then* the Head holds. The Body of the rule is also referred to as the rule goal. A fact is indicated by

$$Head. \tag{3}$$

which is equivalent to

$$\text{Head} :\!- \ true. \tag{4}$$

Prolog can be used to "prove" whether a specific query holds or not. This is performed through *backwards chaining*, which is a backtracking algorithm that usually employs a depth-first search strategy. At the heart of backward chaining is the *unification* operator that matches the arguments of a Prolog goal and a rule head. Unification succeeds if a match is found. Given a query or a goal to prove, backward chaining goes through all the goals and attempts to unify the head of each rule with the goal. If unification succeeds, the rule will be added to a proof stack and the proof proceeds with the body of the rule as the new goal. If a successful proof is found, the variable groundings will be returned in the output. We refer to the rules in the proof stack as the proof trace.

Proposed Model

Our model architecture is depicted in Figure 1. The input is an utterance that has the general form "if $\langle S(X)\rangle$ then $\langle A(X)\rangle$ because $\langle G(X)\rangle$" and the output is a proof trace that represents the system's chain of reasoning. In the first step, the statement goes through a parser that extracts the state, action and goal and converts them to logical terms that consist of a predicate name and a set of arguments. For example, the utterance "wake me up early" is converted to wake(me, early). In the second step, CORGI attempts to prove the goal by submitting a query to its Prolog engine. If

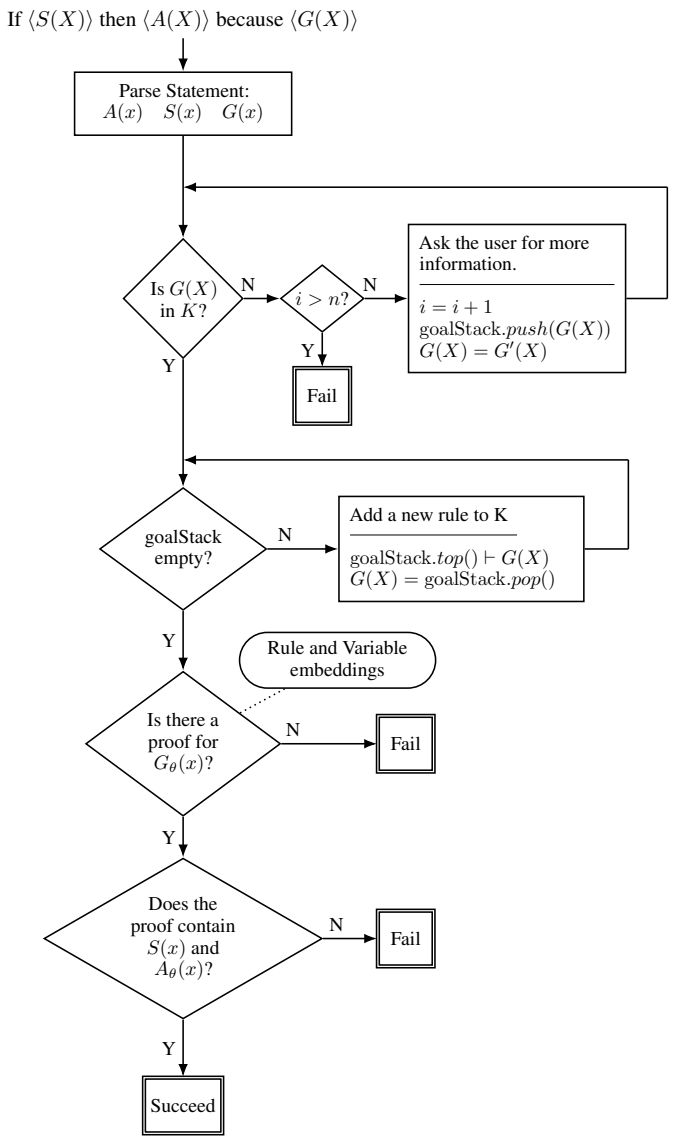

Figure 1: Model flowchart. goalStack and the loop variable i are initialized to empty and 0 respectively.

the proof succeeds, the model moves on to the next statement. If the proof fails, CORGI initiates a dialog with the user and tries to extract the missing knowledge required to prove the goal, from the user. The result of this user feedback loop is knowledge base completion as well as reasoning for the input statement.

It is worth noting that using the because statement is important since if we only rely on the if-then statement for finding the chain of reasoning, will not allow us to choose a relevant path among several returned paths. To be more clear, in the snow example, if we only have the if-then statement without because, if we do not know the because statement we will not be able to indicate for example the days that we need to wake up the user.

We noted that a parser converts natural language utterances to logical representations. Due to the variations of natural language, semantically similar statements could get mapped into different logical forms which is unwanted. for example, "make sure I am awake early morning" vs. "wake me up early morning" which will be parsed into awake(i,early_morning) and wake(me, early_morning), respectively. In order to address this, we propose to learn vector representations or embeddings for logical rules and facts so that we can use them in the proof path when we need to decide if a rule or a fact is relevant in a proof.

In order to learn the rule and fact embeddings, we present a learning method that learns a general proving strategy by observing proof traces of successful proofs. In the next section, we present the details of this proposed model. Before that, let us talk about our parser in more depth in the next subsection.

Parsing

The goal of our parser is to extract the state $S(X)$, action $A(X)$ and goal $G(X)$ from the input utterance and convert them to predicates. The parser is built using Spacy (Honnibal and Montani 2017). We implement a relation extraction method that uses Spacy's builtin dependency parser. The language model that we used is the en_coref_lg$-3.0.0$ released by Hugging face [2]. The predicate name is typically the sentence verb or the sentence root. The predicate's argument are the subject, objects, named entities and noun chunks extracted by Spacy. The output of the relation extractor will be matched against the knowledge base through rule-based mechanisms including string matching to decide weather the parsed logical form exists in the knowledge base. in the future we will use the learned rule embeddings to match the parsed predicate against the knowledge base to make it more robust. If a match is found, the parser re-orders the arguments to match the order of the arguments of the predicate retrieved from the knowledge base. This re-ordering is done through a type coercion method. In order to do type coercion, we use the types released by Allen AI in the Aristo tuple KB v1.03 Mar 2017 Release (Dalvi Mishra, Tandon, and Clark 2017) and have added more entries to it to cover more nouns. The released types file is a dictionary that maps different nouns to their types. For example, doctor is of type person and Tuesday is of type date. If no match is found, the parsed predicate will be kept as is and CORGI will try to find a reasoning link between the new predicate and its existing knowledge base through conversation.

We would like to note that we refrained from using a grammar parser particularly because we wanted to be able to have open-domain discussions with the users and save them the startup time of learning the system's language. An interesting outcome of this is that the system will learn to adapt to the user's language over time since the background knowledge will be accumulated through user interactions and as a result it will be adapted to that user. A negative effect, however, is that if the parser makes a mistake, error will propagate onto the system's future knowledge. We are

[2] https://github.com/huggingface/neuralcoref-models/releases/download/en_coref_lg-3.0.0/en_coref_lg-3.0.0.tar.gz

planning to look into this further in our future work.

Rule Embeddings

Our system uses an embeddings-based approach to enable it to handle the variations typical to natural language settings. state $S(X)$, action $A(X)$ and goal $G(X)$ are all extracted directly from user utterances, and may not exactly match anything in the knowledge base.

The inference algorithm we use for extracting a relevant proof of $G(X)$ is backward chaining. Given a goal to prove, backward chaining searches all the rules and facts in the knowledge base to find the ones whose head unify with the goal. If unification succeeds for a rule, the algorithm moves on to prove the statements in the body of the rule in the same manner. If unification with a fact succeeds the proof terminates with a success. Since our rules and facts are extracted from natural language statements, there is a high chance that there is variation in the arguments and predicate names. For example, unification of awake(i,early_morning) and wake(me, early_morning) will fail using traditional backward chaining. Therefore, we propose a "soft unification" strategy to deal with these variations.

Our soft unification algorithm learns embeddings for the prolog terms and uses vector similarity to make a decision about the success or failure of unification. In order to learn these embeddings, we collect proof traces and maximize the probability of extracting the correct proof for each query by walking down the proof path and maximizing the probability of choosing the correct rule next and making the correct variable groundings if any. Rocktäschel and Riedel (2017) and Weber et al. (2019) also propose a soft unification method, but they present soft "AND" and "OR" operations and propose a differentiable unification operator. Our method is an adaptation of Reed and De Freitas where the authors propose using program execution traces to learn to execute programs. Before we proceed we would like to note that we will represent scalars with lowercase letters, vectors with bold lowercase letters and matrices with bold uppercase letters in what follows. We have one embedding matrix for the rules and facts denoted by $\mathbf{M}^{\text{rule}} \in \mathbb{R}^{n_1 \times m_1}$ where n_1 is number of rules and facts and m_1 is the embedding dimension. Row i in $\mathbf{M}^{\text{rule}}$ corresponds to the i^{th} rule/fact in the knowledge base. We also have a variable embedding matrix denoted by $\mathbf{M}^{\text{var}} \in \mathbb{R}^{n_2 \times m_2}$ where n_2 is the number of all the atoms and variables in the knowledge base and m_2 is the variable embedding dimension. Each variable and atom will have a corresponding row in the variable embedding matrix. Our program is type coerced, therefore the variable names are associated with their types. For example we have the predicate alarm(Person,Time) in the knowledge base. In the next subsection we present the learning algorithm for learning these embeddings and present the details of the model.

Learning

In this section, we present the model that takes as input a query to prove and returns a proof that consists of a sequence of rules and variable bindings. This model is adapted from Reed and De Freitas (2015)'s neural program interpreter that learns to execute programs and is trained on program execution traces. In this paper, we learn to complete proofs by training a model using proof traces.

The model's core consist of an LSTM network whose hidden state indicates the next rule in the proof trace and a proof termination probability, given a query as input and a feed forward network that makes variable binding decisions. The model is fully supervised by the proof trace of a sample query given in a depth-first-traversal order from left to right (Figure 2). The trace is sequentially input to the model in the traversal order. In step t of the proof, the model's input is $\epsilon_t^{inp} = \big(\mathbf{q}_t, \mathbf{r}_t, (\mathbf{v}_t^1, \ldots, \mathbf{v}_t^\ell)\big)$ where $\mathbf{q}_t$ represents the query and is computed by feeding the predicate name of the query into a character RNN, $\mathbf{r}_t$ represents the rules in the parent and the left sister nodes in the proof trace. In order to compute $\mathbf{r}_t$ we concatenate the embeddings of the parent and sister nodes which are looked up from the rule embedding matrix $\mathbf{M}^{rule}$. The arguments of the goal query are presented in $(\mathbf{v}_t^1, \ldots, \mathbf{v}_t^\ell)$ where ℓ is the arity of the goal predicate. Each $\mathbf{v}_t^i$ for $i \in [0, \ell]$, is looked up from the embedding matrix $\mathbf{M}^{var}$. The output of the model in step t is $\epsilon_t^{out} = \big(c_t, \mathbf{r}_{t+1}, (\mathbf{v}_{t+1}^1, \ldots, \mathbf{v}_{t+1}^\ell)\big)$ and is computed through the following equations

$$\mathbf{s}_t = f_{enc}(\mathbf{q}_t) \tag{5}$$

$$\mathbf{h}_t = f_{lstm}(\mathbf{s}_t, \mathbf{r}_t, \mathbf{h}_{t-1}) \tag{6}$$

$$c_t = f_{end}(\mathbf{h}_t), \tag{7}$$

$$\mathbf{r}_{t+1} = f_{rule}(\mathbf{h}_t) \tag{8}$$

$$\mathbf{v}_{t+1}^i = f_{var}(\mathbf{v}_t^i), \tag{9}$$

where $\mathbf{v}_{t+1}^i$ is a probability vector over all the variables and atoms for the i^{th} argument, and $\mathbf{r}_{t+1}$ is a probability vector over all the rules and facts and c_t is a scalar probability of terminating the proof at step t. f_{enc}, f_{end}, f_{rule} and f_{var} are feed forward networks with two fully connected layers, and f_{lstm} is an LSTM network. The trainable parameters of the model are the parameters of the feed forward neural networks, the LSTM network, the character RNN that embeds $\mathbf{q}_t$ and the rule and variable embedding matrices $\mathbf{M}^{rule}$ and $\mathbf{M}^{var}$.

Our model is trained end-to-end. In order to train the model parameters and the embeddings we maximize the negative log likelihood probability given below

$$\boldsymbol{\theta}^* = argmax_{\,\boldsymbol{\theta}} \sum_{\epsilon^{out}, \epsilon^{in}} \log(P(\epsilon^{out}|\epsilon^{in}; \boldsymbol{\theta})) \tag{10}$$

where the summation is over all the proof traces in the training set and $\boldsymbol{\theta}$ is the trainable parameters of the model. We have

$$\log(P(\epsilon^{out}|\epsilon^{in}; \boldsymbol{\theta})) = \sum_{t=1}^{T} \log P(\epsilon_t^{out}|\epsilon_1^{in} \ldots \epsilon_{t-1}^{in}; \boldsymbol{\theta}) \tag{11}$$

$$\begin{aligned} \log P(\epsilon_t^{out}|\epsilon_1^{in} \ldots \epsilon_{t-1}^{in}; \boldsymbol{\theta}) &= \log P(\epsilon_t^{out}|\epsilon_{t-1}^{in}; \boldsymbol{\theta}) \qquad (12) \\ &= \log P(c_t|\mathbf{h}_t) + \\ &\quad \log P(\mathbf{r}_{t+1}|\mathbf{h}_t) + \\ &\quad \log \sum_i P(\mathbf{v}_{t+1}^i|\mathbf{v}_t^i) \qquad (13) \end{aligned}$$

Where the probabilities in Equation (13) are given in Equations (7), (8) and (9).

Algorithm 1 Inference Algorithm

Input: Goal $G(X)$, $\mathbf{M^{rule}}$, $\mathbf{M^{var}}$, Model parameters, threshold T_1, T_2, k
Output: Proof P

```
r_0 ← 0                                   ▷ 0 is a vector of 0s
P = PROVE(G(X), r_0, [])
function PROVE(Q, r_t, stack)
    embed Q using the character RNN to obtain q_t
    input q_t and r_t to the model and compute r_{t+1} (Equation (8))
    compute c_t (Equation (7))
    {R^1 ... R^k} ← From M^rule retrieve k entries corresponding to the top k entries of r_{t+1}
    for i ∈ [0, k] do
        SU ← SOFT_UNIFY(Q, head(R^i))
        if SU == False then
            continue to i + 1
        else
            if c_t > T_2 then
                return stack
            add R^i to stack
            PROVE(Body(R^i))              ▷ Prove the body of R^i
    return stack

function SOFT_UNIFY(G, H)
    Use M^var for variable unification and compute cosine similarity S_1
    if S_1 > T_1 then
        return True
    else
        return False
```

Inference

We use the learned rule and variable embeddings and use cosine similarity to decide if variable unification succeeds or fails in each step. If the cosine similarity is above a certain threshold then unification succeeds otherwise it fails.

Our inference algorithm is based on backward chaining. In each step t of the proof, given a query Q, we compute $\mathbf{q}_t$ using the trained character RNN and $\mathbf{r}_t$ to compute $\mathbf{r}_{t+1}$. Then we choose k entries of $\mathbf{M}^{rule}$ corresponding to the top k entries of $\mathbf{r}_{t+1}$ as candidates for the next proof trace. For each rule in the top k rules, we attempt to do variable unification by computing the cosine similarity between the variables of Q and the variables of the rule's head. If unification succeeds we move to prove the body of that rule. Otherwise, we move to the next rule. Refer to Algorithm 1 for more details.

Datasets

We collected two sets of if/then/because statements from lab members and colleagues. The first set contains 83 statements targeted at LIA-compatible domains (email, calendar, maps, alarms, and weather). The second set contains 77 open-domain statements. 81% of the statements over both sets qualify as "if ⟨state⟩ then ⟨action⟩ because ⟨goal⟩" statements. The remaining 19% differ in the categorization of the *because* clause (see Table 1); common alternate clause types included anti-goals ("...because I don't want to be late"), modifications of the state or action ("If my flight is from 2am to 4am then book me a supershuttle because it will be difficult to find ubers"), or conjunctions including at least one non-goal type. For training the rule embeddings and the parameters of the soft logical reasoning engine we use sPyrolog (Weber et al. 2019) to extract proof traces from our knowledge base.

Domain	*if* clause	*then* clause	*because* clause	Size
LIA-compatible	state	action	goal	76
LIA-compatible	state	action	anti-goal	3
LIA-compatible	state	action	modifier	2
LIA-compatible	state	action	conjunction	2
SUM				83
Open domain	state	action	goal	55
Open domain	state	action	anti-goal	4
Open domain	state	action	modifier	12
Open domain	state	action	conjunction	6
SUM				77

Table 1: Types of if/then/because statements collected from a small pool of human subjects. LIA-compatible domain includes email, calendar, maps, alarms, and weather.

Experimental Results

In this section, we provide the details of the model training and our experimental setup for testing the performance of CORGI. Our rule embeddings were trained on proof traces collected from our knowledge base. The proof traces we collected by passing hand generated queries to sPyrolog. The variable embedding matrix is initialized with GloVe embeddings (Pennington, Socher, and Manning 2014) and the rule embedding matrix is initialized randomly. The embedding dimension of the variables is 300 and the embedding dimension of the rules is 256. Our neural model is implemented in PyTorch (Paszke et al. 2017).

Our knowledge base was a small hand crafted commonsense knowledge represented in terms of if-then rules, also knows as horn clauses. This knowledge base includes general information about time and LIA-related contexts such as setting alarms and notifications, emails and so on as well as open domain commonsense knowledge about day-to-day activities. This knowledge base consists of a total number of 234 facts and rules.

We use a small subset of the if/then/because statements of Table 1 and run a user study to assess the effectiveness of the user's feedback in proving the statements' because goals. 14 people participated in our study and out of the 14 people, 3 were experts that were closely familiar with CORGI and its capabilities. We would like to note that the participants were not necessarily the people from whom we collected the dataset in Table 1. Although it would be best to interact with the same user that inputs the statement, we argue that it is reasonable to use other humans for the explanations. First, most of these statements require day-to-day commonsense knowledge, therefore most humans will be able to give the

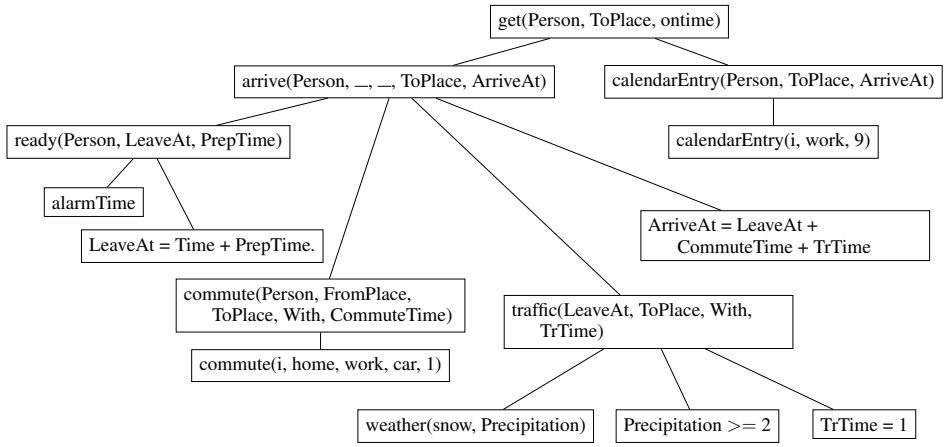

Figure 2: Sample proof tree for the utterance, "I get to work on time." Proof traversal is depth-first from left to right.

Success
If it's going to rain in the afternoon then remind me to bring an umbrella because I want to remain dry.
How do I know if "I remain dry"?
If I have my umbrella.
How do I know if "I have my umbrella"?
If you remind me to bring an umbrella.
Okay, I will perform "remind me to bring an umbrella" in order to achieve "I remain dry".

Failure
If it's going to rain in the afternoon then remind me to bring an umbrella because I want to remain dry.
How do I know if "I remain dry"?
If I have my umbrella.
How do I know if "I have my umbrella"?
If it's in my office.
How do I know if "it's in my office"?
...

Figure 3: Sample dialogs from the user study. System responses are noted in italics.

required commonsense knowledge to the system. For example, some are as simple as having an umbrella to not get wet (Figure 3) Second, we would like to be able to assess the performance of the model on a fixed dataset for comparison purposes.

The results of our user study is given in Table 2. We compare the performance of the model against two scenarios. Scenario no-feedback is when we do not ask the users for any feedback and attempt to prove the because goal against the knowledge base. The oracle scenario is when our soft unification has access to perfect rules so that soft unification always succeeds when it should and fails when it should.

Discussion

Our system's ability to successfully interpret an if/then/because statement was heavily dependent on whether the user knew how to give the system what it needed, not necessarily what it asked for. A sample pair of success and failure dialogs is in figure 3. The goal of this study was to indicate if users are able to give any useful feedback that can complete the commonsense knowledge of CORGI. The results indicate that it is indeed possible to rely on users to complete the underlying knowledge of the system. However, we noticed that in order to make the responses effective, we need to improve our parser which is what we intend to do in our future work. Moreover, in the future we wish to extend CORGI to handle conjunction statements. Furthermore, the rule embeddings are not currently updated as the user interacts with the model. Therefore, in the future we plan to investigate neural models that adaptively update and add embeddings for new facts and rules on the fly, as the user interacts with the system.

Model	Novice User	Expert User
No-feedback	0	0
CORGI	16/88=18.18%	8/24= 33.33%
Oracle	20/77 = 25.97%	10/21 = 47.61%

Table 2: Comparison of the performance of the model under different scenarios. In the no-feedback scenario no conversation with the user is established. In the Oracle scenario, soft unification is 100% accurate.

Conclusions

In this paper, we introduced CORGI (COmmon-sense ReasoninG by Instruction) that performs reasoning by initiating a conversation with a user. CORGI has access to a small knowledge base of commonsense facts and completes this knowledge base through time as she interacts with the user. We define common-sense reasoning as the process of finding a chain of reasoning in a logic program given an if-then-because statement. We show that obtaining the because statement is crucial in extracting a relevant chain of reasoning given an if-then statement

References

Azaria, A.; Krishnamurthy, J.; and Mitchell, T. M. 2016. Instructable intelligent personal agent. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Bosselut, A.; Rashkin, H.; Sap, M.; Malaviya, C.; Celikyilmaz, A.; and Choi, Y. 2019. Comet: Commonsense transformers for automatic knowledge graph construction. *arXiv preprint arXiv:1906.05317*.

Chen, D.; Bai, Y.; Zhao, W.; Ament, S.; Gregoire, J. M.; and Gomes, C. P. 2019. Deep reasoning networks: Thinking fast and slow. *arXiv preprint arXiv:1906.00855*.

Cohen, W. W. 2016. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

Colmerauer, A. 1990. An introduction to prolog iii. In *Computational Logic*, 37–79. Springer.

Dalvi Mishra, B.; Tandon, N.; and Clark, P. 2017. Domain-targeted, high precision knowledge extraction. *Transactions of the Association for Computational Linguistics* 5:233–246.

Evans, R., and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61:1–64.

Hixon, B.; Clark, P.; and Hajishirzi, H. 2015. Learning knowledge graphs for question answering through conversational dialog. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 851–861.

Honnibal, M., and Montani, I. 2017. spacy 2: Natural language understanding with bloom embeddings. *Convolutional Neural Networks and Incremental Parsing*.

Hu, H.; Yarats, D.; Gong, Q.; Tian, Y.; and Lewis, M. 2019. Hierarchical decision making by generating and following natural language instructions. *arXiv preprint arXiv:1906.00744*.

Hudson, D. A., and Manning, C. D. 2018. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*.

Justice, L. M.; Pence, K. L.; and Beckman, A. 2005. *Scaffolding with Storybooks: A Guide for Enhancing Young Children's Language and Literacy Achievement*. International Reading Association.

Labutov, I.; Srivastava, S.; and Mitchell, T. 2018. Lia: A natural language programmable personal assistant. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 145–150.

Li, T. J.-J.; Azaria, A.; and Myers, B. A. 2017. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 6038–6049. ACM.

Li, T. J.-J.; Li, Y.; Chen, F.; and Myers, B. A. 2017. Programming iot devices by demonstration using mobile apps. In *International Symposium on End User Development*, 3–17. Springer.

Li, T. J.-J.; Labutov, I.; Li, X. N.; Zhang, X.; Shi, W.; Ding, W.; Mitchell, T. M.; and Myers, B. A. 2018. Appinite: A multimodal interface for specifying data descriptions in programming by demonstration using natural language instructions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 105–114. IEEE.

Liang, X.; Hu, Z.; Zhang, H.; Lin, L.; and Xing, E. P. 2018. Symbolic graph reasoning meets convolutions. In *Advances in Neural Information Processing Systems*, 1853–1863.

Luketina, J.; Nardelli, N.; Farquhar, G.; Foerster, J.; Andreas, J.; Grefenstette, E.; Whiteson, S.; and Rocktäschel, T. 2019. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Rashkin, H.; Sap, M.; Allaway, E.; Smith, N. A.; and Choi, Y. 2018. Event2mind: Commonsense inference on events, intents, and reactions. *arXiv preprint arXiv:1805.06939*.

Reed, S., and De Freitas, N. 2015. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*.

Rocktäschel, T., and Riedel, S. 2017. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, 3788–3800.

Saeidi, M.; Bartolo, M.; Lewis, P.; Singh, S.; Rocktäschel, T.; Sheldon, M.; Bouchard, G.; and Riedel, S. 2018. Interpretation of natural language rules in conversational machine reading. *arXiv preprint arXiv:1809.01494*.

Sap, M.; LeBras, R.; Allaway, E.; Bhagavatula, C.; Lourie, N.; Rashkin, H.; Roof, B.; Smith, N. A.; and Choi, Y. 2018. Atomic: An atlas of machine commonsense for if-then reasoning. *arXiv preprint arXiv:1811.00146*.

Serafini, L., and Garcez, A. d. 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*.

Srivastava, S.; Azaria, A.; and Mitchell, T. M. 2017. Parsing natural language conversations using contextual cues. In *IJCAI*, 4089–4095.

Srivastava, S.; Labutov, I.; and Mitchell, T. 2017. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, 1527–1536.

Srivastava, S. 2018. *Teaching Machines to Classify from Natural Language Interactions*. Ph.D. Dissertation, Samsung Electronics.

Wang, P.-W.; Donti, P. L.; Wilder, B.; and Kolter, Z. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*.

Wang, S. I.; Liang, P.; and Manning, C. D. 2016. Learning language games through interaction. *arXiv preprint arXiv:1606.02447*.

Weber, L.; Minervini, P.; Münchmeyer, J.; Leser, U.; and Rocktäschel, T. 2019. Nlprolog: Reasoning with weak unification for question answering in natural language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, ACL 2019, Florence, Italy, Volume 1: Long Papers*, volume 57. ACL (Association for Computational Linguistics).

Wu, C.; Liu, J.; Wang, X.; and Dong, X. 2018. Chain of reasoning for visual question answering. In *Advances in Neural Information Processing Systems*, 275–285.

Yi, K.; Wu, J.; Gan, C.; Torralba, A.; Kohli, P.; and Tenenbaum, J. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, 1031–1042.

Zhou, Z.-H. 2019. Abductive learning: towards bridging machine learning and logical reasoning. *Science China Information Sciences* 62(7):76101.