Can BERT Conduct Logical Reasoning? On the Difficulty of Learning to Reason from Data

Anonymous ACL submission

Abstract

Logical reasoning is needed in a wide range of NLP tasks. In this work, we seek to answer one research question: can we train a BERT model to solve logical reasoning problems written in natural language? We study this problem on a confined problem space and train a BERT model on randomly drawn data. However, we report a rather surprising finding: even if BERT achieves nearly perfect accuracy on the test data, it only learns an incorrect and partial reasoning function; further investigation shows that the behaviour of the model (i.e., the learned partial reasoning function) is unreasonably sensitive to the training data. Our work reveals the difficulty of learning to reason from data and shows that near-perfect performance on randomly drawn data is not a sufficient indicator of models' ability to conduct logical reasoning.

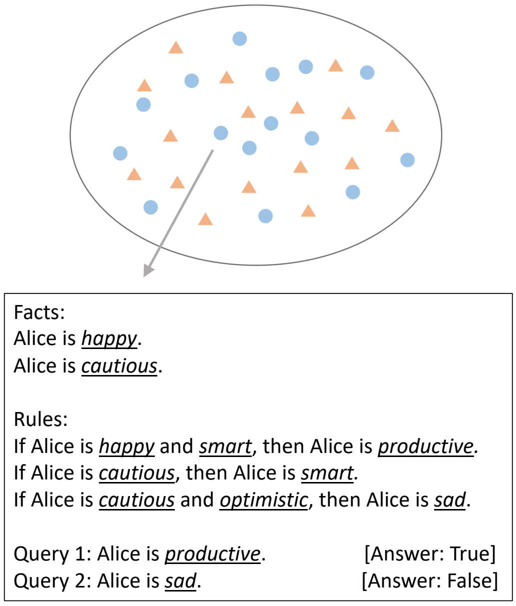

Figure 1: A visualization of our problem setting. The large circle denotes a confined problem space consisting of logical reasoning problems. The dots and the triangles represent two independently sampled sets of examples. The lower-half of the figure shows an example of a logical reasoning problem sampled from the problem space.

1 Introduction

Logical reasoning is needed in a wide range of NLP tasks including natural language inference (NLI) (Williams et al., 2018; Bowman et al., 2015), question answering (QA) (Rajpurkar et al., 2016; Yang et al., 2018) and common-sense reasoning (Zellers et al., 2018; Talmor et al., 2019). The ability to draw conclusions based on given facts and rules, as shown in Figure 1, is fundamental to solving these tasks[1]. Though NLP models, empowered by the Transformer neural architecture (Vaswani et al., 2017), can achieve high performance on task-specific datasets (Devlin et al., 2019), it is unclear whether they are able to reason logically over the input as humans do. A research question naturally arises: *can neural networks be trained to conduct logical reasoning presented in English?*

Prior work (Liu et al., 2020; Tian et al., 2021) answers this question by training and testing NLP models on datasets consisting of logical reasoning problems written in natural language (Figure 1). Since neural models have limited capacity (e.g. the computational complexity of neural models is polynomial in input length), it is unreasonable to expect them to solve arbitrarily complex logical reasoning problems (e.g. 3-SAT) (Cook, 1971). A common practice (Johnson et al., 2017; Sinha et al., 2019) is to train and test the models on a *confined problem space*, where we limit the difficulty of the problems by controlling the number/complexity of the facts and rules in each example. Besides, since it is infeasible to enumerate all examples in the problem space, the models are trained on datasets of reasonable size by randomly drawing examples from the problem space. Following this procedure,

[1] A.k.a., deductive reasoning. In this paper, we do not consider inductive reasoning, where the rules need to be learned.

Clark et al. (2020) suggests that neural models can be trained to conduct logical reasoning by showing that they achieve high performance on such randomly generated datasets.

In this work, we argue that performing well on a set of examples randomly sampled from the problem space does not entail that the model is conducting logical reasoning. We first note that, given a problem space, there can be multiple ways to sample examples (Sec. 3.1); each sampling method implicitly defines a *probability distribution* over the problem space and can be used to generate different *datasets* conforming to this distribution. Thus, if a model is conducting logical reasoning, it should perform consistently well on datasets sampled by different algorithms, i.e., on the whole intended problem space. This expectation of "reasoning ability" is reasonable: algorithms such as forward chaining (Russell and Norvig, 2002) can solve logical reasoning problems regardless of how the test set is generated, and it is natural to expect the same from a "reasoning" neural model.

We show that neural models, even when trained to a nearly perfect accuracy on randomly generated data, still fail to generalize over the entire reasoning problem space and thus do not learn to reason. We investigate this issue in a controlled problem space called SimpleLogic (Sec. 2). We first show that BERT has sufficient capacity to solve SimpleLogic by proving that there exists a parametrization for BERT that can solve all instances in SimpleLogic (Sec. 2.2). Then, to test whether BERT can *learn* such reasoning ability from data, we present two sampling approaches to generate datasets for SimpleLogic: Rule-Priority (RP) and Label-Priority (LP). In RP we first sample the facts and rules, which then naturally determines the True/False labels of the predicates, while in LP we first determine the predicate labels and then randomly generate rules and facts consistent with the pre-assigned labels (see Fig. 3 for an illustration). Both sampling approaches are intuitive and simple, covering the whole problem space of SimpleLogic. Therefore, as illustrated in Figure 1, we expect a model trained on data generated by either RP or LP (denoted by the dots and triangles, respectively) to generalize to the whole problem space.

However, we observe that even though the BERT model has no difficulty reaching near-perfect performance on data generated by RP, it fails catastrophically when tested on LP (and vice versa) (Sec. 3). Furthermore, we find that the BERT model is unreasonably sensitive to the training distribution, in that the model behaviour changes significantly as the sampling method that generates the training data changes (Sec. 4).

The results indicate that BERT learns an incomplete reasoning function that does not generalize to the whole problem space. The learned function is also specific to its training distribution, which is undesirable as the correct reasoning function is defined by the problem space rather than the training distribution.

Our study unveils the difficulty of learning to reason from data and we illustrate that such generalization failure is inherently different from the typical generalization errors in NLP tasks (Sec. 5). Our finding leads to one major implication: in contrast to common practice, showing near-perfect performance on a randomly drawn testset is not a sufficient indicator of the logic reasoning ability of a model. Source code and data for reproducing the experiments will be released upon acceptance.

2 SimpleLogic: A Simple Logical Reasoning Problem Space

We define *SimpleLogic*, a class of logical reasoning problems based on propositional logic. We use SimpleLogic as a controlled testbed for testing neural models' ability to conduct logical reasoning.

SimpleLogic only contains deductive reasoning examples. To simplify the problem, we remove language variance by representing the reasoning problems in a templated language and constrain its complexity (e.g., examples have limited input lengths, number of predicates, and proof tree depths).

Solving SimpleLogic does not require significant computational capacity. We show that a popular pre-trained language model BERT (Devlin et al., 2019)[2] has more than enough computational capacity to solve SimpleLogic. That is, there exists a parameterization of BERT that solves SimpleLogic with 100% accuracy (Sec. 2.2).

2.1 Problem Space Definition

Before we present the formal definition for SimpleLogic, we introduce some basics for propositional logic. In general, reasoning in propositional logic

[2]BERT is one of the most popular language model backbones for NLP downstream models. In this paper, we use BERT as a running example and our conclusion can be naturally extended to other Transformer-based NLP models.

is NP-complete; hence, we only consider propositional reasoning with *definite clauses*. A definite clause in propositional logic is a *rule* of the form $A_1 \wedge A_2 \wedge \cdots \wedge A_n \rightarrow B$, where A_is and B are *predicates* that take values in "True" or "False"; we refer to the left hand side of a rule as its *body* and the right hand side as its *head*. In particular, a definite clause is called a *fact* if its body is empty (i.e. $n = 0$). A *propositional theory* (with only definite clauses) T is a set of rules and facts, and we say a predicate Q *can be proved* from T if either (1) Q is given in T as a fact or (2) $A_1 \wedge \cdots \wedge A_n \rightarrow Q$ is given in T as a rule where A_is can be proved.

Each example in SimpleLogic is a propositional reasoning problem that only involves definite clauses. In particular, each example is a tuple (*facts*, *rules*, *query*, *label*) where (1) *facts* is a list of predicates that are known to be True, (2) *rules* is a list of rules represented as definite clauses, (3) *query* is a single predicate, and (4) *label* is either True or False, denoting whether the query predicate can be proved from *facts* and *rules*. Figure 1 shows such an example. Additionally, we enforce some simple constraints to control the difficulty of the problems. For each example in SimpleLogic, we require that:

- the number of predicates ($pred\_num$) that appear in facts, rules and query ranges from 5 to 30, and all predicates are sampled from a fixed vocabulary containing 150 adjectives such as "happy" and "complicated";
- the number of rules ($rule\_num$) ranges from 0 to $4 \times pred\_num$, and the body of each rule contains 1 to 3 predicates; i.e. $A_1 \wedge \cdots \wedge A_n \rightarrow B$ with $n > 3$ is not allowed;
- the number of facts ($fact\_num$) ranges from 0 to $pred\_num$;
- the reasoning depth[3] required to solve an example ranges from 0 to 6.

We use a simple template to encode examples in SimpleLogic as natural language input. For example, we use "*Alice is X.*" to represent the fact that X is True; we use "*A and B, C.*" to represent the rule $A \wedge B \rightarrow C$; we use "*Query: Alice is Q.*" to represent the query predicate Q. Then we concatenate *facts*, *rules* and *query* as *[CLS] facts. rules [SEP] query [SEP]* and supplement it to BERT to predict the correct *label*.

[3]For a query with label *True*, its reasoning depth is given by the depth of the shallowest proof tree; for a query with label *False*, its reasoning depth is the maximum depth of the shallowest failing branch in all *possible* proof trees.

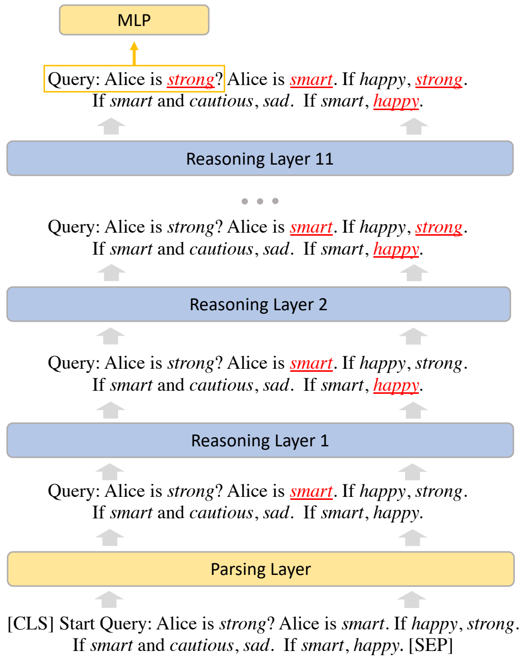

Figure 2: A visualization of a BERT-base model that simulates the forward-chaining algorithm. The first layer is a parsing layer, converting text input into the desired format. The underlined predicates are proven or known as facts. Each reasoning layer performs one step of forward-chaining. For example, for Reasoning Layer 2, given that "happy" has been proven, it applies the rule "If happy, (then) strong" to prove the predicate *strong*, which is underlined in the output of this layer.

2.2 BERT Has Enough Capacity to Solve SimpleLogic

In the following, we show that BERT has enough capacity to solve all examples in SimpleLogic. In particular, we explicitly construct a parameterization for BERT such that the fixed-parameter model solves all problem instances in SimlpleLogic. Note that we only prove the existence of such a parameterization, but do not discuss whether such a parameterization can be learned from sampled data until Sec. 3.

Theorem 1 *For BERT with n layers, there exists a set of parameters such that the model can correctly solve any reasoning problem in SimpleLogic that requires $\leq n - 2$ steps of reasoning.*

We prove this theorem by construction. We construct a fixed set of parameters for BERT to simulate the forward-chaining algorithm. Here we show a sketch of the proof, and refer readers to the Appendix for the full proof. As illustrated in Figure 2, our construction solves a logical reasoning example in a layer-by-layer fashion. The 1st layer of

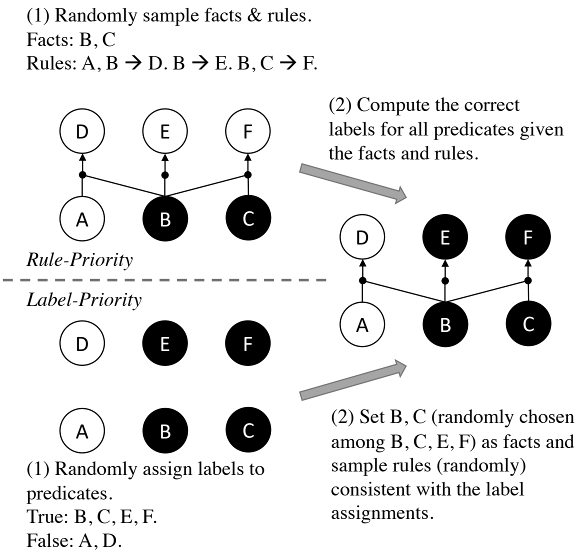

Figure 3: An illustration of a logical reasoning problem (right) in SimpleLogic being sampled by Rule-Priority (RP) and Label-Priority (LP), respectively. Predicates with label *True* are denoted by filled circles.

BERT parses the input sequence into the desired format. Layer 2 to layer 10 are responsible for simulating the forward chaining algorithm: each layer performs one step of reasoning, updating the True/False label for predicates. The last layer also performs one step of reasoning, while implicitly checking if the query predicate has been proven and propagating the result to the first token. The parameters are the same across all layers except for the Parsing Layer (1st layer).

We implemented the construction in PyTorch, following the exact architecture of the BERT-base model. The "constructed BERT" solves all problems in SimpleLogic of reasoning depth ≤ 10 with 100% accuracy, using only a small proportion of the parameters of BERT.

3 BERT Fails to Learn to Solve SimpleLogic

Next, we study whether it is possible to train a neural model (e.g., BERT) to reason on SimpleLogic. We follow the standard approach (Clark et al., 2020): we randomly sample examples from the problem space and train BERT on a large amount of sampled data. We consider two natural ways to sample data from SimpleLogic. We expect if a model can learn to reason, the model should be able to solve examples generated by any sampling methods once it is trained.

3.1 Sampling Examples from SimpleLogic

We consider two intuitive ways of sampling the examples. (1) Rule-Priority (RP): we first randomly generate rules and facts, which then determine the label of each predicate (Algorithm 1). (2) Label-Priority (LP): we first randomly assign a True/False label to each predicate and then randomly sample some rules and facts that are consistent with the pre-assigned labels (Algorithm 2). Figure 3 shows an example illustrating the two sampling methods. RP is fully general and directly follows from the definition of SimpleLogic. However, there is no simple way to control certain properties of the generated examples such as the number of proof trees (see Sec. 4 for more examples). On the other hand, LP makes it easier to control the properties of the generated examples (in Sec. 4, we utilize LP to generate a suite of test sets to probe model behaviours).

3.2 BERT Trained on Random Data Cannot Generalize

Following the two sampling regimes described above, we randomly sample two sets of examples from SimpleLogic: for each reasoning depth from 0 to 6, we sample $80k$ examples from SimpleLogic via Algorithm RP (LP) and aggregate them as dataset RP (LP), which contains $560k$ examples in total. We then split it as training/validation/test set. We train a BERT-base model (Devlin et al., 2019) on RP and LP, respectively. We train for 20 epochs with a learning rate of 4×10^{-5}, a warmup ratio of 0.05, and a batch size of 64. Training takes less than 2 days on 4 GPUs.

BERT performs well on the training distribution. The first and last rows of Table 1 show the test accuracy when the test and train examples are sampled by the same algorithm (e.g., for Row 1, the model is trained on the training set of RP and tested on the test set of RP). In such scenarios, the models can achieve near-perfect performance similar to the observations in prior work (Clark et al., 2020). Both of our sampling algorithms are general in the sense that every instance in SimpleLogic has a probability to be sampled in either RP or LP. Thus, the intuition is that models achieving near-perfect performance on such a general dataset should emulate the correct reasoning function.

BERT fails to generalize. However, at the same time, we observe a rather counter-intuitive finding: the test accuracy drops significantly when the

Algorithm 1 *Rule-Priority* (RP)

$pred\_num \sim U[5, 30]$
$preds \leftarrow Sample(vocab, pred\_num)$
$rule\_num \sim U[0, 4 * pred\_num]$
$rules \leftarrow$ empty array
while size of $rules < rule\_num$ **do**
  $body\_num \sim U[1, 3]$
  $body \leftarrow Sample(preds, body\_num)$
  $tail \leftarrow Sample(preds, 1)$
  add $body \rightarrow tail$ to $rules$
end while
$fact\_num \sim U[0, pred\_num]$
$facts \leftarrow Sample(preds, fact\_num)$
$query \leftarrow Sample(preds, 1)$
Compute $label$ via forward-chaining.
return $(facts, rules, query, label)$

Algorithm 2 *Label-Priority* (LP)

$pred\_num \sim U[5, 30]$
$preds \leftarrow Sample(vocab, pred\_num)$
$rule\_num \sim U[0, 4 * pred\_num]$
set $l \sim U[1, pred\_num/2]$ and group $preds$ into l layers
for predicate p in layer $1 \leq i \leq l$ **do**
  $q \sim U[0, 1]$
  assign label q to predicate p
  if $i > 1$ **then**
    $k \sim U[1, 3]$
    $cand \leftarrow$ nodes in layer $i - 1$ with label $= q$
    $body \leftarrow Sample(cand, k)$
    add $body \rightarrow p$ to $rules$
  end if
end for
while size of $rules < rule\_num$ **do**
  $body\_num \sim U[1, 3]$
  $body \leftarrow Sample(preds, body\_num)$
  $tail \leftarrow Sample(preds, 1)$
  add $body \rightarrow tail$ to $rules$ unless $tail$ has label 0 and all predicates in $body$ has label 1.
end while
$facts \leftarrow$ predicates in layer 1 with label $= 1$
$query \leftarrow Sample(preds, 1)$
$label \leftarrow$ pre-assigned label for $query$
return $(facts, rules, query, label)$

Figure 4: Two sampling algorithms Rule-Priority and Label-Priority. $Sample(X, k)$ returns a random subset from X of size k. $U[X, Y]$ denotes the uniform distribution over the integers between X and Y.

train and test examples are sampled via different algorithms. Specifically, as shown in the second and third row of Table 1, the BERT model trained on RP fails drastically on LP, and vice versa. As illustrated in Figure 1, if a model performs well on the dots (RP), it is expected that it performs well on the triangles (LP). Such failure to generalize to the whole problem space indicates BERT is not conducting logical reasoning, even if we train the model on the data sampled by a general algorithm. A subsequent question naturally arise: is this simply because the two algorithms are complementary? If we train the model on data sampled by both algorithms, can the model learn to reason?

Train	Test	0	1	2	3	4	5	6
RP	RP	99.8	99.9	99.5	99.2	98.8	97.5	96.4
	LP	99.2	99.9	99.0	91.9	84.5	69.7	52.8
LP	RP	100.0	96.0	79.3	71.2	70.1	71.5	74.8
	LP	100.0	100.0	99.9	99.9	99.7	99.7	99.0

Table 1: BERT trained on RP achieves almost perfect accuracy on its test set; however the accuracy drops significantly when it's tested on LP (vice versa).

Test	0	1	2	3	4	5	6
RP&LP	99.9	99.9	99.7	99.5	99.4	99.0	97.1
LP*	97.2	97.2	93.6	82.7	71.4	58.4	53.6

Table 2: BERT trained on a mixture over RP and LP fails on LP*, a test set that differs from LP only slightly.

Training on both RP and LP is not enough. We train BERT on the mixture of RP and LP, and BERT again achieves nearly perfect test accuracy. Can we now conclude that BERT is conducting reasoning? We slightly tweak the sampling algorithm of LP by increasing the expected number of alternative proof trees to generate LP*, which is a special case of the LP3 test set, to be introduced in Sec. 4. Unfortunately, we observe that the model performance drops significantly on LP* (Table 2). The accuracy drops to 53.6% for the reasoning depth of 6 on LP*, even if the model achieves over 96% in validation. Such a result resembles what we observed in Table 1, where the model fails to generalize outside of its training distribution, even if we are enriching the training distributions with different sampling methods. In fact, we find no evidence that consistently enriching the training distribution will bring a transformative change such that the model can learn to reason, as we cannot enumerate all distributions (see a discussion in Sec. 5.1).

Discussion. The experiments above reveal a pattern of failure: if we train the model on one data distribution, it fails almost inevitably on a different distribution. In another word, the model seems to be emulating an incorrect "reasoning function" specific to its training distribution. The results imply that for logical reasoning problems, the test accuracy on a dataset generated by one particular sampling algorithm should not be used as the sole indicator of models' reasoning ability.

4 BERT is Sensitive to Training Distribution

As shown in the previous section, though the BERT model achieves near-perfect test accuracy on the data distribution it is trained on, it fails catastrophically on the others. BERT does not learn the algorithm that allows it to solve all problems from SimpleLogic. In this section, we study how sensitive the model behaviour is to the training distribution changes. Intuitively, if a model is emulating the correct reasoning function, an insignificant change to the training distribution (e.g., slightly increasing the average fact number) should not incur large changes in model behaviours.

We first create a suite of similar training distributions by slightly tweaking the parameters of the sampling algorithm. We train different models on these training distributions and analyze their behaviours. As it is hard to fully characterize the behaviour of a black-box model, we use the performance on a test suite as a proxy for model behaviour; each test set in the suite is created to probe a particular aspect of model behavior. We show that even as we slightly change the training distribution (e.g., the rule number distribution), the performance of the BERT model on the test suite changes significantly.

4.1 Experiment Setup

Variants of training sets. We first describe how we tweak the parameters of Algorithm 1 to obtain a suite of slightly different training distributions. Such changes are insignificant as the resulting distributions still cover the whole problem space. The sampling algorithm RP is mainly governed by two parameters, $fact\_num$ and $rule\_num$ and we introduce shifts in the underlying distribution of RP by tweaking the distribution of $fact\_num$ and $rule\_num$. We propose to adapt Algorithm 1 in the following way: in line 3 and 11 of Algorithm 1, instead of sampling $fact\_num$ and $rule\_num$ from uniform distributions, we sample them from the binomial distributions $B(pred\_num, fact\_p)$ and $B(4 * pred\_num, rule\_p)$, respectively; here $fact\_p$ and $rule\_p$ are two hyper-parameters we use to change the mean of $fact\_num$ and $rule\_num$. For the modified algorithm, we enumerate $fact\_p$ from $(0.1, 0.2, 0.3, 0.4)$ to construct training set F-0.1, F-0.2, F-0.3 and F-0.4, and $rule\_p$ from $(0.3, 0.4, 0.5, 0.6)$ to construct training set R-0.3, R-0.4, R-0.5 and R-0.6. As we change $fact\_p$ and $rule\_p$, the overall nature of the sampling algorithm stays the same and guarantees that all examples in SimpleLogic can be sampled with a positive probability. Here, training set F-0.2 and R-0.5 are the same.

Dataset	Property
LP1	Label Priority
LP2	Proof trees form disjoint cycles
LP3	Alternative proof trees
LP4	Abundant rules but cycle-free
LP5	Symmetric labels
LP6	Unique proof tree

Table 3: Each dataset focuses on different properties.

A suite of test sets. While it is hard to fully characterize the behavior of a black-box model, we can detect changes in model behavior by a probing method: if a model's performance on some test sets changes, its behavior changes. We use variants of Algorithm 2 to generate a suite of test sets, each with a different focus in probing. For example, when generating random rules in Algorithm 2 (line 17 - 22), by adding more constraints, we can prioritize rules that could create alternative paths thus increasing the expected number of alternative proof trees[4] in the generated examples. Table 3 briefly describes some high-level properties for each dataset (e.g., examples from LP3 have more alternative proof trees). Regarding the specific sampling algorithms that generate the test sets, we refer readers to the code for further details.

4.2 Results and Analysis

Table 4 and Table 5 shows the performance of models trained with different rule number distributions ($rule\_p$) and different fact number distributions ($fact\_p$), respectively. We report the mean accuracy on examples of reasoning depths $4 - 6$.

BERT still fails. Our first observation is that for all models trained on RP variants, their performance drops significantly when tested on the LP variants, echoing our findings in Sec. 3.2. This

[4]A proof tree is a directed graph consisting of the rules and facts contributing to the proof of a predicate with label *True* (see Fig. 3. There could be multiple proof trees for one example (i.e., different ways to prove the query) and we call them alternative proof trees.

Model	R-0.3	R-0.4	R-0.5	R-0.6
validation	98.4	98.1	98.2	98.2
LP1 (0.58)	72.1	75.4	80.1	**80.9**
LP2 (0.24)	71.9	**74.8**	68.2	66.1
LP3 (0.58)	74.4	83.4	85.9	**88.1**
LP4 (0.71)	73.9	87.0	84.9	**94.3**
LP5 (0.21)	75.5	**82.8**	74.4	76.8
LP6 (0.21)	**79.1**	76.8	78.2	73.6

Table 4: The performance on LP variants when $rule\_p$ changes in training. The value in parentheses following the test set show the statistics of $rule\_p$ in the test set.

Model	F-0.1	F-0.2	F-0.3	F-0.4
validation	98.2	98.2	98.2	98.3
LP1 (0.12)	73.9	80.1	**83.9**	83.0
LP2 (0.10)	58.7	68.2	**74.5**	72.8
LP3 (0.06)	82.5	**85.9**	77.7	80.0
LP4 (0.08)	83.6	84.9	**92.5**	92.2
LP5 (0.08)	71.5	74.4	79.8	**79.9**
LP6 (0.08)	74.2	**78.2**	70.3	76.5

Table 5: The performance on LP variants when $fact\_p$ changes in training. The value in parentheses following the test set show the statistics of $fact\_p$ in the test set.

again verifies that the failure of BERT to generalize is systematic and persistent.

There is no "optimal" parameters. When we change the rule number distribution ($rule\_p$), the models' performance on the test sets fluctuates and there is not a single $rule\_p$ value that achieves the highest performance on all test sets. For example, on LP1, $rule\_p = 0.6$ achieves the best performance but on LP3, $rule\_p = 0.4$ does. Such observation also holds when we change the fact number distribution. The high sensitivity to the training distribution is undesirable, as the difference between training distribution is small and we keep all training distribution general.

BERT behavior is sensitive to distribution shifts. The fluctuations on the test set as the training set changes are seemingly bizarre. In some cases, the performance changes on the LP variants can be intuitively explained. For example, in Table 4, LP4 has greater rule numbers and intuitively, if a training distribution has more rules, the model trained on it performs better. Indeed, $rule\_p = 0.6$ achieves the best performance on LP4. In other cases, it is hard to explain the performance changes with the intuitive "rule of thumb". For example, in Table 5, larger $fact\_p$ implies more alternative proof trees; however, in LP3, where examples tend to have many alternative proof trees, the best performance is achieved when $fact\_p = 0.2$ rather than $fact\_p = 0.4$. In fact, we find that such "bizarre" fluctuations are attributed to the inherent issue of learning to reason from data, and a deeper analysis is provided in Sec. 5.

5 Discussions

In this work, we show that even when BERT has more than enough capacity to solve SimpleLogic, it is difficult for BERT to learn the ability to solve SimpleLogic from data. In this section, we discuss the reason and implication of such phenomena.

5.1 Why BERT Fails to Generalize to the Whole Problem Space?

It is a common observation that neural models may not generalize to more difficult examples (Sinha et al., 2019). In the context of logical reasoning, the difficulty of an example is conventionally defined as its reasoning depth, and models do not generalize well to examples of reasoning depth larger than the training examples (Clark et al., 2020). Thus, to ensure good performance during test time, we need to sample training examples that are at least as "difficult" as potential test examples. However, in our experiments, we observe that even though the test and train examples are "equally difficult" in the conventional sense (e.g., they have the same reasoning depths), the model still generalizes poorly.

We provide an explanation for this atypical form of generalization failure: the neural model has a different notion of difficulty compared to humans. Specifically, we posit that (1) the difficulty of an example for a neural model is characterized by multiple difficulty factors beyond the reasoning depth; (2) the factors contributing to an example's difficulty could be hard to identify, imperceptible, or go against human intuitions. For example, we note one such "hard to identify" factor: the number of alternative proof trees (see a definition in Sec. 4.1). An alternative proof tree with a greater depth than the optimal tree could mislead the model to take more steps to arrive at the correct answer. If a model is trained on examples with few alternative proof trees, it may generalize poorly to examples with a large number of proof trees. In Sec. 4, LP3 is created to test models' ability to handle alternative proof trees and all models fail on LP3.

The difficulty factors for the model could also go against human intuition: examples that appear easy for a human could be hard for the model. For example, LP2 is simple extremely simple for humans (Sec. 4.1): every example contains exactly one proof tree, which is a simple cycle with no alternative paths. However, all models struggle to solve LP2, even though sometimes they can solve other seemingly more challenging examples with complex proof trees.

In fact, LP2 - LP6 from Sec. 4 are created based on our guess about what kind of examples could be considered "difficult" by the model. Nevertheless, they are not an exhaustive enumeration of all of difficulty factors, as many factors could be compositional or nonsensical, and thus intractable to enumerate. Thus, it is almost futile to try to sample training examples that are "difficult" enough for the test time, as we do not know the model's definition of "difficulty".

5.2 Beyond Logical Reasoning

In this section, we discuss the implications of our findings beyond the scope of logical reasoning.

Dataset bias. Prior work finds that neural models fail to generalize when training data contain obvious dataset biases or shortcuts from the data annotation or collection process (Gururangan et al., 2018).For example, due to shortcuts in the NLI datasets (McCoy et al., 2019), NLI models may rely on the fallible lexical overlap heuristic: a premise entails all hypotheses constructed from words in the premise. The generalization failure presented in this paper can be viewed as a novel type of dataset bias: the training data for logical reasoning contain no annotation or data collection biases in the traditional sense; however, the training data distribution does indeed allow for the existence of an incorrect function that performs well on the training distribution but fails on the whole problem space. In other words, there exist intricate and intractable "shortcuts" or "biases" in our training data (Sec. 5.1). We hope our findings deepen the understanding of dataset bias beyond annotation artifacts.

Using synthetic data. It is a common practice to use synthetic data as a proxy for a certain class of problems (Johnson et al., 2017; Weston et al., 2016). In this case, a random sampling algorithm is used to draw examples from the defined problem space to form a dataset. A model's ability to solve the class of problems is determined by the its test performance on the sampled dataset. However, as the random sampling algorithm appears general (i.e., every example has a positive probability to be sampled by it), it is often neglected whether the test performance truly reflects the model's ability to generalize to the whole problem space. Our results show that caution should be taken and the high test performance could be misleading.

6 Related Work

A great proportion of NLP tasks require logical reasoning. Prior work contextualizes the problem of logical reasoning by proposing reasoning-dependent datasets and studies solving the tasks with neural models (Johnson et al., 2017; Sinha et al., 2019; Yu et al., 2020; Liu et al., 2020; Tian et al., 2021). However, most studies focus on solving a single task, and the datasets either are designed for a specific domain (Johnson et al., 2017; Sinha et al., 2019), or have confounding factors such as language variance (Yu et al., 2020). They can not be used to strictly or comprehensively study the logical reasoning abilities of models. In contrast, we propose SimpleLogic, a simple yet general scenario of logical reasoning, to analyze the model reasoning ability. Furthermore, in contrast to the common practice, we show performance on a randomly drawn testset is not sufficient to be an indicator of logic reasoning ability of a model.

Another line studies leveraging deep neural models to solve pure logical problems. For examples, SAT (Selsam et al., 2019), maxSAT (Wang et al., 2019), temporal logical problems (Hahn et al., 2021), DNF counting (Crouse et al., 2019), logical reasoning by learning the embedding of logical formula (Crouse et al., 2019; Abdelaziz et al., 2020) and mathematical problems (Saxton et al., 2019; Lample and Charton, 2020). In this work, we focus on deductive reasoning, which is a general and fundamental reasoning problem. Clark et al. (2020) conducts a similar study to show that models can be trained to reason over language, while we observe the difficulty of learning to reason from data.

7 Conclusion

In this work, we study whether BERT can be trained to conduct logical reasoning in a confined problem space called SimpleLogic. Even though we show that the BERT model has enough capacity to solve SimpleLogic perfectly, it fails to learn the correct reasoning function from examples that are randomly sampled from the problem space. We call for caution in future work and show that the high performance on one validation dataset does not entail generalization to the whole problem space.

References

Ibrahim Abdelaziz, Veronika Thost, Maxwell Crouse, and Achille Fokoue. 2020. An experimental study of formula embeddings for automated theorem proving in first-order logic. *CoRR*, abs/2002.00423.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*. The Association for Computational Linguistics.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *IJCAI*. ijcai.org.

Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM.

Maxwell Crouse, Ibrahim Abdelaziz, Cristina Cornelio, Veronika Thost, Lingfei Wu, Kenneth D. Forbus, and Achille Fokoue. 2019. Improving graph neural network representations of logical formulae with subgraph pooling. *CoRR*, abs/1911.06904.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*. Association for Computational Linguistics.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *NAACL-HLT (2)*. Association for Computational Linguistics.

Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. 2021. Teaching temporal logics to neural networks. In *ICLR*. OpenReview.net.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2017. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*. IEEE Computer Society.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics. In *ICLR*. OpenReview.net.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In *IJCAI*. ijcai.org.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, pages 2383–2392. The Association for Computational Linguistics.

Stuart Russell and Peter Norvig. 2002. Artificial intelligence: a modern approach.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *ICLR (Poster)*. OpenReview.net.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *ICLR (Poster)*. OpenReview.net.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. 2019. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *EMNLP/IJCNLP (1)*, pages 4505–4514. Association for Computational Linguistics.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL-HLT (1)*, pages 4149–4158. Association for Computational Linguistics.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through logicnli. In *EMNLP (1)*. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, Proceedings of Machine Learning Research. PMLR.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomás Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *ICLR (Poster)*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, pages 2369–2380. Association for Computational Linguistics.

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. Reclor: A reading comprehension dataset requiring logical reasoning. In *ICLR*. OpenReview.net.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *EMNLP*. Association for Computational Linguistics.

A Construction Proof of Theorem 1

We prove theorem 1 by construction: in N-layer BERT model, we take the first layer as parsing layer, the last layer as output layer and the rest layers as forward chaining reasoning layer. Basically, in the parsing layer we preprocess the natural language input. In forward chaining reasoning layers, the model iteratively broadcast the RHSs to all LHSs, and check the left hand side (LHS) of each rule and update the status of the right hand side (RHS). Here we introduce the general idea of the construction, and we will release the source code for the detailed parameters assignments.

A.1 Pre-processing Parameters Construction

Predicate Signature For each predicate P, we generate its signature $Sign_P$, which is a 60-dimensional unit vector, satisfying that for two different predicates P_1, P_2, $Sign_{P_1} \cdot Sign_{P_2} < 0.5$. We can randomly generate those vectors and check until the constraints are satisfied. Empirically it takes no more than 200 trials.

Meaningful Vector In parsing layer, we process the natural language inputs as multiple “meaningful vectors". The meaningful vectors are stored in form of $L_A||L_B||L_C||R||0^{512}$, representing a rule $L_A \wedge L_B \wedge L_C \rightarrow R$. Each segment L_A, L_B, L_C, R has 64 dimensions, representing a predicate or a always True/False dummy predicate. For each predicate P, the first 63 dimensions, denoted as P^{sign}, form the signature of the predicate, and the last dimension is a boolean variable, denoted as P^v. The following information is converted into meaningful vectors:

- Rule $LHS \rightarrow RHS$: if the LHS has less than 3 predicates, we make it up by adding always True dummy predicate(s), and then encode it into meaningful vector, stored in the separating token follows the rule. In addition, for each predicate P in LHS, we encode a dummy meaningful vector as $False \rightarrow P$ and store it in the encoding of P. This operation makes sure that every predicate in the input sentence occurs at least once in RHS among all meaningful vectors. We will see the purpose later.
- Fact P: we represent it by a rule $True \rightarrow P$, and then encode it into meaningful vector and store it in the embedding of the separating token follows the fact.
- Query Q: we represent it by a rule $Q \rightarrow Q$, encode and store it in the [CLS] token at beginning.

Hence, in the embedding, some positions are encoded by meaningful vectors. For the rest positions, we use zero vectors as their embeddings.

A.2 Forward Chaining Parameters Construction

Generally, to simulate the forward chaining algorithm, we use the attention process to globally broadcast the true value in RHSs to LHSs, and use the MLP layers to do local inference for each rule from the LHS to the RHS.

In attention process, for each meaningful vector, the predicates in LHS look to the RHS of others (including itself). If a RHS has the same signature as the current predicate, the boolean value of the RHS is added to the boolean value of the current predicate. Specifically, we construct three heads. We denote $Q_i^{(k)}$ to stand for the query vector of the i-th token of the k-th attention head. For a meaningful vector written as $L_A||L_B||L_C||R||0^{512}$,

$$
\begin{aligned}
&Q_i^{(1)} = L_A^{sign}||\frac{1}{4}, Q_i^{(2)} = L_B^{sign}||\frac{1}{4}, Q_i^{(1)} = L_C^{sign}||\frac{1}{4} \\
&K_i^{(1)} = \beta R, K_i^{(2)} = \beta R, K_i^{(3)} = \beta R \\
&V_i^{(1)} = 0^{63}||R^v, V_i^{(2)} = 0^{63}||R^v, V_i^{(3)} = 0^{63}||R^v.
\end{aligned}
$$

Here β is a pre-defined constant. The attention weight to a different predicate is at most $\frac{3\beta}{4}$, while the attention weight to the same predicate is at least β, and the predicate with positive boolean value has even larger ($\frac{5\beta}{4}$) attention weight. Thus, with a large enough constant β, we are able to make the attention distribution peaky. Theoretically, when $\beta > 300 \ln 10$, we can guarantee that the attention result

$$
Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V
$$

satisfies that the value is in the range of $[0.8, 1.0]$ if the predicate on LHS is boardcasted by some

RHS with true value, otherwise it is in the range of $[0, 0.2]$.

This attention results are added to the original vectors by the skipped connection. After that, we use the two-layer MLP to do the local inference in each meaningful vector. Specifically, we set

$$
\begin{aligned}
10[&ReLU(L_A^v + L_B^v + L_C^v - 2.3) \\
&-ReLU(L_A^v + L_B^v + L_C^v - 2.4)]
\end{aligned}
$$

as the updated R^v. Thus, $R^v = 1$ if and only if all the boolean values in LHS are true, otherwise $R^v = 0$. We also set L_A^v, L_B^v, L_C^v as 0 for the next round of inference.

A.3 Output Layer Parameters Construction

In output layer, we take out the boolean value of the RHS of the meaningful vector in [CLS] token.